Mise en correspondance et gestion de la cohérence de modèles hétérogènes évolutifs

Mahmoud El Hamlaoui

THÈSE

En vue de l'obtention du

DOCTORAT DE L'UNIVERSITÉ DE TOULOUSE

Délivré par :

Université Toulouse - Jean Jaurès

Cotutelle internationale avec "Université Mohammed V de Rabat"

Présentée et soutenue par :
Mahmoud EL HAMLAOUI
le vendredi 18 septembre 2015

Titre :
Mise en correspondance et gestion de la cohérence de modèles hétérogènes évolutifs

---

École doctorale et discipline ou spécialité :
ED MITT : Domaine STIC : Réseaux, Télécoms, Systèmes et Architecture

Unité de recherche :
Institut de Recherche en Informatique de Toulouse (IRIT)

Directeur/trice(s) de Thèse :

Bernard COULETTE, Professeur, Université Toulouse Jean Jaurès
Mahmoud NASSAR, Professeur Habilité, Université Mohammed V de Rabat

Jury :

Mireille BLAY-FORNARINO, Professeur, Université Nice Sophia Antipolis
Antoine BEUGNARD, Professeur, Télécom Bretagne
Bouchaïb BOUNABAT, Professeur, Université Mohammed V de Rabat
Sophie EBERSOLD, Maître de Conférences, Université Toulouse Jean Jaurès
Adil ANWAR, Professeur Habilité, Université Mohammed V de Rabat

«لْتُقَ يِبَ زِ يضدز ًمًلًع»
[114 : سوية طه]

«Et dis : Ô Seigneur augmente mon savoir»
[sôurat Taha : 114]

Du plus profond de mon cœur, je dédie ce travail à mes parents pour lesquels j'exprime mon amour et ma gratitude pour leur sacrifice et leur soutien moral. Que dieu vous garde.

Remerciements

"Gratitude is when memory is stored in the heart and not in the mind"

— *Lionel Hampton*

Avant tout, un grand merci à Dieu de m'avoir donné la force d'entamer et d'achever cette thèse.

Je remercie Madame Mireille Blay-Fornarino, Monsieur Antoine Beugnard et Monsieur Bouchaib Bounabat d'avoir rapporté sur ce manuscrit, ainsi que pour leurs critiques et suggestions constructives ayant permis de l'améliorer. Je les remercie également d'avoir fait partie de mon jury de thèse.

Mes remerciements vont aussi à mes directeurs de thèse, Monsieur Bernard Coulette et Monsieur Mahmoud Nassar pour leur support scientifique et personnel tout au long de ma thèse. Je tiens aussi à leur assurer ma profonde gratitude pour m'avoir initié à la recherche.

Un grand merci à Madame Sophie Ebersold et Monsieur Adil Anwar pour m'avoir encadré, pour leur disponibilité, leur investissement et les nombreux conseils qu'ils m'ont prodigué.

Mes remerciments vont également à tous les membres de l'équipe MACAO et à ceux de l'équipe SIME pour les discussions intéréssantes et tous les bons moments partagés ensemble.

Je tiens également à remercier les membres du département Mathématique-Informatique de l'UT2J et ceux du département Informatique de l'IUT de Blagnac pour leurs conseils, leurs qualités humaines et leurs hospitabilité.

J'adresse aussi mes vifs remerciments aux thésards que j'ai croisés quasi-quotidennment et avec qui j'ai eu l'occasion de partager des repas et pauses cafés sympathiques. Je parle particulièrement d'Anaïs et d'Erasme qui ont évéillé en moi l'amour de l'anthropologie et de la géographie. Merci pour votre amitié et votre encouragments.

Ce travail n'aurait pas abouti sans le soutien tout d'abord de mes parents sans lesquels tout cela n'aurait pas été possible. Merci pour vos sacrifices. Merci aussi aux prunelles de mes yeux, mes trois petites sœurs : Fayna, Farah et Khadija. Je suis navré de m'être trop éloigné de vous. Je vous souhaite un avenir plein de succès et de bonheur. Mes plus profonds remerciements vont aussi à Yousra ma meilleure amie pour son soutien sans faille pendant ce long périple dans les bons comme dans les pires moments et qui connait maintenant mon domaine de recherche aussi bien que moi (ou presque).

Je tiens aussi à témoigner de ma profonde reconnsaissance à mon parrain Taoufiq Dkaki auprès duquel je cherchais conseils dans les moments de doute et aussi pour les inoubliables moments passés ensemble aux berges des rivières à m'initier à la pêche. Cependant l'éléve n'a toujours pas réussi à surpasser le maître.

Mes remerciments ne peuvent se clôturer sans une pensée particulière à mes amis, les trois mousquetaires (Rahma, Eric, Adel), Imed, Yassine, Ilyasse, Jalal, Ahmed, Haytam, Mouad, Youssef, Elhoucine, Simy, Qamsane, Tarick, Amine, Adil, Omar, Taisha, Eloise, Charlotte, Sabrina, Mailis, Sandrine, Célia, Morgane, Jacob, Innocent, Aymerick, Arnaud, Martin, Ely, Georges, aux membres de l'antenne jeune d'Amnesty, à l'équipe du basket-ball. La liste est très longue, mes excuses pour ceux que j'ai oubliés.

Résumé

Pour permettre la compréhension et la manipulation d'un système complexe, le découpage en parties séparées est nécessaire. En Ingénierie Dirigée par les Modèles (ou *Model Driven Engineering*), ces parties sont représentées par des modèles, que nous qualifions de modèles partiels, dans la mesure où ils sont focalisés sur des domaines métiers distincts. Dans ce contexte de multi-modélisation, ces modèles sont dits hétérogènes quand ils sont décrits dans des langages de modélisation distincts dédiés à différents domaines métiers : DSML (*Domain Specific Modeling language*). La compréhension et l'exploitation efficace des connaissances relatives à un tel système supposent la construction d'un modèle global représentant son fonctionnement.

La création du modèle global requiert l'identification des correspondances existant entre les éléments des différents modèles partiels. Dans la pratique, ces correspondances sont soit incomplètement identifiées, soit insuffisamment formalisées pour être maintenues lorsque les modèles partiels évoluent. Ceci limite leur utilisation et ne permet pas de les exploiter pleinement lors de la construction du modèle global ou du traitement de l'évolution des modèles partiels.

L'apport de cette thèse est double.

La première contribution est celle d'un processus permettant la création d'une vue globale du système par l'intermédiaire d'une composition fondée sur la mise en correspondance des modèles partiels. Les correspondances identifiées entres les éléments des modèles se basent sur des types de relations instanciées à partir d'un métamodèle de correspondance. Ce dernier est extensible (selon les spécificités du domaine d'application considéré) et permet de supporter les concepts relatifs à ce domaine. Les correspondances sont d'abord identifiées entre les méta-éléments des métamodèles respectifs des modèles partiels. Les correspondances entre les éléments de modèles sont ensuite obtenues par un mécanisme de raffinement, supporté par un langage d'expression sémantique ad hoc : SED (*Semantic Expression DSL*). La composition est dite «virtuelle» dans la mesure où les éléments figurant dans une correspondance ne sont que des références aux éléments appartenant aux modèles partiels. De ce fait, les modèles interconnectés par ces correspondances forment un modèle global virtuel.

La seconde contribution est relative au maintien de la cohérence des modèles partiels et du modèle global. En effet, les modèles évoluant dans le temps, le changement d'un élément ou de plusieurs éléments participant à l'expression des correspondances, peut entrainer l'incohérence du modèle global. Pour maintenir la cohérence du modèle global, nous proposons un second processus permettant tout d'abord d'identifier automatiquement les changements réalisés ainsi que leurs classifications et leurs répercussions sur les éléments de modèles concernés. Par la suite, les différents cycles sont gérés à l'aide de l'expert puis une liste de changements est générée en fonction de la stratégie choisie et des coefficients de pondération. Enfin, le traitement des changements est réalisé de façon semi-automatique.

Ce travail a été concrétisé par le développement d'un outil support nommé HMCS (*Heterogeneous Matching and Consistency management Suite*), basé sur la plateforme Eclipse.

L'approche a été validée et illustrée à travers un cas d'étude portant sur la gestion du Service d'Urgence d'un hôpital. Ce travail a été mené en collaboration avec le CHU de Montpellier.

Mots clés : MDE, multi-modélisation, DSML, hétérogénéité de modèles, correspondance, composition, traitement de l'évolution.

Abstract

To understand and manipulate a complex system, it is necessary to apply the separation of concerns and produce separate parts. In Model Driven Engineering (MDE), these parts are represented by models qualified as partial models. In this context of multi-modeling, these models are called heterogeneous when they are described in separate modeling languages dedicated to different business domains: DSML (*Domain Specific Modeling Language*).

Global model creation requires identifying existing correspondences between the elements of the partial models. However, in practice these correspondences are either incompletely identified or not sufficiently formalized to be maintained when the partial models evolve. This restricts their use and does not allow to fully exploit them for building the global model or for treating partial models evolution.

The contribution of this thesis is twofold.

The first contribution deals with a process for creating a global view of the system by means of a composition based on partial models matching. Identified correspondences between models elements are based on types of relationship instantiated from a metamodel of correspondences. This latter is extensible, depending on the considered application domain, and allows supporting the concepts related to this domain. Correspondences are firstly identified between meta-elements belonging to metamodels of the respective partial models. Correspondences between model elements are then obtained by a refinement mechanism, supported by an ad hoc Semantic Expression language: SED (*Semantic Expression DSL*). The composition is called "virtual" since elements represented in a correspondence are only references to elements belonging to partial models. Therefore, models interconnected by this correspondences form a virtual global model.

The second contribution relates the consistency of the global model. Indeed, as models evolve over time, changing one or several elements involved in a correspondence, may cause the inconsistency of the global model. To maintain its consistency, we propose a second process enabling to automatically identify the changes, classify them and treat their impacts on the involved model elements. Management of repercussions is performed semi-automatically by the expert by means of strategies and weights.

This work has been implemented through a support tool named HMCS (Heterogeneous Matching and Consistency management Suite) based on the Eclipse Platform.

The approach has been validated and illustrated through a case study related to the management of a Hospital Emergency Service. This work was led in collaboration with the "CHU of Montpellier".

Keywords: MDE, multi-modeling, DSML, heterogeneous models, correspondences, composition, evolution processing.

Table des Matières

Table des figures

Liste des tableaux

CHAPITRE I. INTRODUCTION GÉNÉRALE

"There are no big problems, there are just a lot of little problems"

— *Henry Ford*

Ce travail de thèse s'est déroulé dans le cadre du projet PHC Volubilis ADAPROC (2011-2013) et d'une thèse en cotutelle entre l'université de Toulouse 2 Jean-Jaurès et l'université Mohammed V de Rabat.

Les équipes de recherche impliquées dans le présent travail de thèse sont les suivantes :

- Equipe MACAO (Modèles, Aspects, Composants, Agilité et prOcessus) au sein du laboratoire IRIT (Institut de Recherche en Informatique de Toulouse), France,
- Equipe IMS (Ingénierie des Modèles et Systèmes) au sein du laboratoire SIME (Systèmes d'Information Mobiles et Embarqués), Rabat, Maroc.

I.1. Contexte et Motivations

Aujourd'hui le développement de systèmes logiciels dépend d'un ensemble de langages, d'outils et de processus qui sont généralement utilisés séparément par des experts en modélisation travaillant sur différentes dimensions ou aspects du système. En outre, les experts sont souvent situés dans des zones géographiques éloignées, comme c'est le cas dans le développement collaboratif distribué, ce qui ne facilite pas leur coopération. Avec ce large éventail de langages et d'outils, le défi des experts est d'assurer une cohérence globale pour avoir une construction correcte du système (Bhave et al., 2011).

Plusieurs approches ont été développées pour faire face à la modélisation des systèmes complexes. L'une des plus efficaces et des plus largement utilisées dans l'industrie consiste à élaborer des modèles distincts qui correspondent aux différents points de vue du système (Hilliard, 2001) (Koning et Van Vliet, 2006) (Boulanger et al., 2010). Cette approche ─ dite de multi-modélisation (Boronat et al., 2009) ─ est utilisée en MDE (Model driven Engineering) en plaçant les modèles au cœur du développement des systèmes. Elle permet d'utiliser des langages correspondant à différents métiers permettant ainsi aux concepteurs de se concentrer sur les différentes parties du système de façon indépendante.

Ce souci n'est pas uniquement un problème industriel récent, il a été mis en évidence il y a 378 ans par Descartes (Descartes et Gröber, 1905) à travers des règles théoriques d'analyse et de

synthèse pour apprendre à raisonner, sans pour autant trouver une façon (industrielle) de le concrétiser. La première règle d'analyse consiste à décomposer un problème en différents sous-problèmes : «Diviser chacune des difficultés en autant de parcelles qu'il se pourrait et qu'il serait requis pour les mieux résoudre». Quand on décompose un programme, il est facile de raisonner sur des fonctionnalités précises. Actuellement, en MDE, la modélisation d'un système se fait en utilisant des langages de modélisation dédiés, des DSMLs (Domain Specific Modeling Languages) permettant de cibler et de restreindre les diverses préoccupations, de réduire la complexité en termes de modularité et de simplicité d'utilisation, de faciliter la maintenance et ainsi de constituer les différentes vues partielles du système. Dans le domaine de l'aéronautique par exemple, il est d'usage de développer différents modèles correspondant à diverses vues métier : électricien, thermicien, informaticien, mécanicien, etc. La seconde règle édictée par Descartes, dite de synthèse, permet d'assembler les éléments décomposés de l'analyse : «Conduire par ordre mes pensées en commençant par les objets les plus simples et les plus aisés à connaitre, pour monter peu à peu comme par degrés jusqu'à la connaissance des plus composés». La nécessité d'une phase de synthèse s'inscrit dans le besoin de construire le système complexe en assemblant les vues partielles identifiées précédemment pour disposer d'une vue globale du système. Les modèles peuvent donc être construits et testés de façon indépendante et assemblés par la suite dans une solution complète au moment opportun. Etant donné que les modèles sont construits de manière décomposable, ceci permet de donner plus de contrôle sur la façon dont le système évolue au fil du temps.

Dans notre équipe, nous travaillons depuis longtemps sur la thématique de la modélisation multivue. Nous avons ainsi développé une méthodologie d'analyse/conception décentralisée par point de vue traduite par l'élaboration d'un profil UML appelé VUML (Nassar, 2003) et par une démarche associée. Celle-ci passe par une étape de composition (de type fusion) de modèles partiels décrits en UML, afin de produire un modèle décrit dans le profil VUML. Pour automatiser (au moins partiellement) cette fusion de modèles partiels, nous avons appliqué l'approche MDE. Pour cela, nous avons traité la composition de diagrammes UML (plus précisément les diagrammes de classes (Anwar et al., 2010) et les diagrammes d'états-transitions (Lakhrissi, 2010)) comme des transformations prenant en entrée plusieurs modèles partiels conformes à UML et produisant en sortie un modèle conforme à VUML. Un prototype d'outil support a été réalisé en ATL (Jouault et al., 2008), et a été validé sous l'environnement Eclipse (Gronback, 2009).

La principale contrainte de cette approche fondée sur VUML est d'imposer un langage de modélisation unique, à savoir UML. Si cette contrainte peut être acceptable pour la conception centralisée de systèmes purement logiciels, il n'en est pas de même lorsque les systèmes sont plus complexes et nécessitent une conception décentralisée. Aussi avons-nous décidé de nous placer dans le cadre de modèles partiels décrits dans des langages adaptés aux différents domaines «métier» d'un système complexe. L'utilisation de langages dédiés correspond à la pratique industrielle et facilite la réalisation des modèles d'analyse/conception.

Cette problématique de multi-modélisation est partagée par la communauté Génie Logiciel comme en témoigne notamment l'action spécifique GEMOC du GDR GPL (Gpl), et le projet international éponyme (Gemoc, 2012). Elle est indissociable de la notion d'hétérogénéité. Une classification des différents types d'hétérogénéité a été abordée dans (Baudry et al., 2012). Le premier type concerne l'hétérogénéité de systèmes (ex : système ULS : *Ultra-Large-Scale* (Gabriel et al., 2006)). Le deuxième type concerne l'hétérogénéité des plates-formes d'exécution (ex :

machines de simulation). Le troisième type concerne l'hétérogénéité de la modélisation qui traite la coordination des modèles conçus selon des DSMLs différents. Ce dernier type d'hétérogénéité est au cœur de nos travaux actuels.

Nous nous plaçons désormais dans le contexte d'une démarche guidée par les points de vue mais en considérant des modèles décrits dans des DSMLs différents par des acteurs aux domaines métiers distincts. Une solution pour traiter l'hétérogénéité des modèles partiels en utilisant l'approche VUML consisterait, en amont, à transformer chacun d'entre eux en un modèle UML et à appliquer ensuite le profil VUML. Cette solution centralise et optimise le format de données afin d'unifier la représentation des modèles partiels, que l'on peut alors composer. Cette approche, dont le principe est d'utiliser UML comme langage pivot, a été écartée car jugée trop lourde à mettre en œuvre et donc ne passant pas à l'échelle. En effet, il faut dans ce cas tout d'abord réaliser des transformations des modèles partiels hétérogènes vers des modèles UML, ce qui est loin d'être simple selon le degré d'hétérogénéité, mais il faut aussi ré-exécuter ces transformations à la moindre évolution des modèles partiels.

Une deuxième solution consiste à composer (fusionner) les différents modèles partiels sans passer par un modèle pivot. Globalement, les approches mettant en œuvre cette composition possèdent trois inconvénients majeurs liés à l'hétérogénéité des modèles. Le premier inconvénient concerne la structure du métamodèle associé au modèle composé. En effet, il n'y a pas de consensus sur la façon dont il doit être construit : de l'union de tous les méta-éléments provenant des métamodèles des modèle partiels (métamodèle d'entrée) ou de leur intersection. Le deuxième inconvénient concerne la sémantique utilisée pour représenter un élément d'un modèle composé dans la mesure où les modèles partiels se fondent sur différentes sémantiques. Le troisième inconvénient concerne la taille du modèle global (composé) qui risque d'être énorme ce qui rendrait son maintien très difficile.

I.2. Périmètre et principe de notre approche

Notre objectif est de proposer une approche permettant de connecter les modèles partiels (modèles de niveau M1 selon la classification en «pyramide» de l'OMG (OMG, 1989)) − donc sans produire de modèle global physique − en constituant en quelque sorte un modèle « virtuel » sous forme d'un «réseau» de modèles partiels. Cette approche s'inscrit dans le cadre de la multi-modélisation mais il ne serait pas réaliste de traiter des modèles partiels appartenant à n'importe quel domaine métier, et donc potentiellement très éloignés les uns des autres d'un point de vue sémantique. Aussi considérons-nous que le système complexe à modéliser appartient à un domaine (global) d'application donné (exemple : aéronautique, spatial, banque, automobile, etc.) dont le vocabulaire est globalement maitrisable par un expert. Les concepteurs des modèles partiels exercent des métiers différents et produisent des modèles hétérogènes, dans leurs domaines de compétences que nous appelons domaines «métier». Les domaines métiers décrivent les différentes perspectives d'un domaine d'application. Par exemple, pour un système complexe dans le domaine (d'application) de l'aéronautique, les concepteurs produisent les modèles métier suivants : modèle mécanique, modèle électrique, modèle informatique, etc.

Notre approche se présente sous la forme d'un processus général composé lui-même de deux (sous-)processus. Les acteurs impliqués dans le processus global sont les concepteurs des domaines métier, l'expert intégrateur et un outil support. Le premier (sous-)processus consiste à

créer des correspondances inter-modèles (et non intra-modèles) entre les différents modèles partiels (considérés comme des entrées). Le second (sous-)processus traite l'évolution et le maintien de la cohérence du système en s'appuyant sur les correspondances issues du premier processus.

Le processus de mise en correspondance proposé s'appuie sur un métamodèle de correspondance qui est générique, extensible selon le domaine d'application et flexible. Sa flexibilité lui permet de s'adapter en fonction du nombre de modèles et des correspondances inter-modèles à créer qui peuvent être binaires, ou n-aires (n>2). Pour obtenir les correspondances entre les modèles, nous identifions tout d'abord les correspondances entre leurs métamodèles (*High Level Correspondences* - HLC) puis générons de façon semi-automatique les correspondances au niveau modèle (*Low Level Correspondences* - LLC). La transition entre les HLCs et les LLCs est effectuée par un mécanisme de raffinement qui s'appuie sur SED (*Semantic Expression DSL*), un langage d'expression de la sémantique des types de relation que nous proposons pour définir les correspondances. Le modèle de correspondance produit contenant les LLCs, permet d'avoir une vue globale du système par virtualisation. Cela veut dire que les éléments des modèles partiels ne sont pas copiés dans la modèle global mais référencés par un mécanisme de virtualisation présenté dans (Clasen et al., 2011a). Cette façon de composer les modèles promeut notre modèle de correspondance au rang de modèle global virtuel. Ce dernier peut servir à la génération (partielle) du code du système (Moreira et al., 2010), à l'interopérabilité (Bruneliere et al., 2010), aux tests (Baudry et al., 2010), à la gestion de l'évolution (Gray et al., 2006), etc.

L'utilisation du modèle global dans l'évolution est mise en œuvre dans un second sous-processus qui a pour objectif, en exploitant le modèle de correspondance, d'identifier les changements effectués sur les modèles partiels de façon automatique. Les changements sont classés puis les différents impacts potentiels sont identifiés. Les éventuelles influences en cascade cyclique sont identifiées et gérées à l'aide de l'expert. Par la suite, selon la stratégie choisie, une liste de changements est générée et ordonnée. Finalement, les changements sont traités de façon semi-automatique pour garantir le maintien de la cohérence globale des modèles interconnectés. Ce second processus pourrait permettre d'ajouter une autre fonctionnalité au modèle global qui est la gestion de la traçabilité (Aizenbud-Reshef et al., 2006), exploitée lors du maintien de la cohérence. Ceci peut servir par exemple, une fois le modèle global construit, à connaitre, suite aux modifications des spécifications d'un système, les parties du code à modifier et inversement.

Nous avons développé comme preuve de concept un outil support opérationnel sous l'environnement Eclipse appelé HMCS (*Heterogeneous Matching and Consistency management Suite*). Il se présente comme un assistant à la création du modèle global virtuel et au maintien de sa cohérence. Il a été utilisé dans différents cas d'étude et notamment pour la modélisation d'un service d'urgence d'un hôpital, étude effectuée en collaboration avec le CHU de Montpellier.

I.3. Organisation du mémoire

La suite de cette thèse est organisée en sept chapitres. Le Chapitre II introduit l'environnement scientifique sur lequel s'appuie cette thèse. Plus précisément, il introduit les principes généraux de l'ingénierie à base de modèles et présente un tour d'horizon des standards développés. Il décrit aussi succinctement le profil VUML développé par notre équipe et ses

différentes exploitations. Le Chapitre III propose une revue de la littérature du domaine de la thèse selon trois axes. Le premier axe traite les approches de mise en correspondance de modèles. Le deuxième axe aborde les approches de composition de modèles selon un processus unique alors que le troisième axe traite les approches de traitement de l'évolution de modèles. La contribution de la thèse est présentée dans les chapitres IV et V, corroborée par un exemple fil conducteur de gestion de conférences. Le Chapitre IV détaille la première contribution consistant en une approche permettant de créer un modèle global virtuel à partir d'un ensemble de modèles partiels hétérogènes. Le Chapitre V présente la deuxième contribution, à savoir une approche de gestion de la cohérence du modèle global lorsque les modèles partiels évoluent. Le Chapitre VI décrit un prototype implémentant les contributions de cette thèse. Il s'agit de l'outil support HMCS développé autour de la plateforme Eclipse qui assiste l'expert intégrateur lors de l'établissement des correspondances et lors du traitement des impacts des différents changements pour maintenir la cohérence du système. Ce travail de thèse est illustré dans le Chapitre VII par une étude de cas portant sur la modélisation du service d'urgence d'un hôpital. Enfin, nous concluons et présentons nos perspectives de travail dans le Chapitre VIII. Une liste des publications produites pendant cette thèse, ainsi qu'une liste d'acronymes sont fournies à la fin de ce document. Des annexes complètent ce document pour en illustrer certaines parties : l'Annexe A présente l'algorithme de raffinement des relations de correspondance, l'Annexe B illustre en EBNF la grammaire Xtext, et l'Annexe C présente les principales abbréviations.

CHAPITRE II. CONTEXTE

"To a man with a hammer, everything looks like a nail. To a Computer Scientist, everything looks like a language design problem. Languages and compilers are, in their opinion, the only way to drive an idea into practice"
— *David Parnas*

II.1. Introduction

Ce chapitre présente le contexte scientifique dans lequel se situe notre travail. Nous traitons tout d'abord de l'ingénierie à base de modèles puis des travaux de notre équipe qui se situent sur le même axe. Nous utilisons dans ce chapitre les acronymes en anglais.

II.2. Ingénierie à base de modèles : (*Model Based Engineering* - MBE)

Avant d'expliciter le MBE il est nécessaire de définir la notion de modèle. Dans (Bézivin et Gerbé, 2001), un modèle est défini comme une représentation d'un système construit pour un objectif précis. Le modèle doit répondre aux questions que les utilisateurs se posent sur le système qu'il représente. Le MBE est donc la formalisation de l'application des modèles tout au long du cycle de vie d'un projet (Friedenthal et al., 2007). L'accent est mis sur la modélisation du problème plutôt que son codage. Il s'agit de remplacer l'approche centrée sur les documents appliquée par le passé par les ingénieurs systèmes par une approche centrée sur les modèles adoptés par différentes disciplines de l'ingénierie. Le MBE n'est pas une révolution de l'ingénierie, il s'agit simplement d'effectuer les mêmes activités mais cette fois-ci en utilisant les modèles au lieu des documents. Ces derniers pouvant être produits à tout moment à partir des modèles.

Lenny Delligatti (Delligatti, 2013), fait remarquer, dans un webinaire à propos des solutions à base de modèles, que le MBE n'est pas une lampe magique (a *silver bullet* selon ses termes). Il n'élimine pas la nécessité d'un processus d'ingénierie rigoureux. Un travail important doit être fourni en mettant la bonne information dans les modèles en entrée afin d'obtenir de bonnes informations dans les modèles en sortie.

II.2.1. Anthropologie du MD*

Le MBE a donné naissance à un ensemble d'approches dirigées par les modèles à savoir le MDE (*Model Driven Engenerring*) ainsi que son application à l'ingénierie logicielle connue sous le nom de MDSE (*Model Driven Software Engineering*), le MDEE (Model Driven Entreprise Engineering), le MDPLE (Model Driven Product Line), le MDD (*Model Driven Developpment*), le MDA (*Model Driven Architecture*), etc. Etant donné que tous les MD*E sont des variantes du MDE, nous ne discuterons dans les sections suivantes que des initiatives représentées dans la Figure II-1 à savoir : MDE, MDD et MDA.

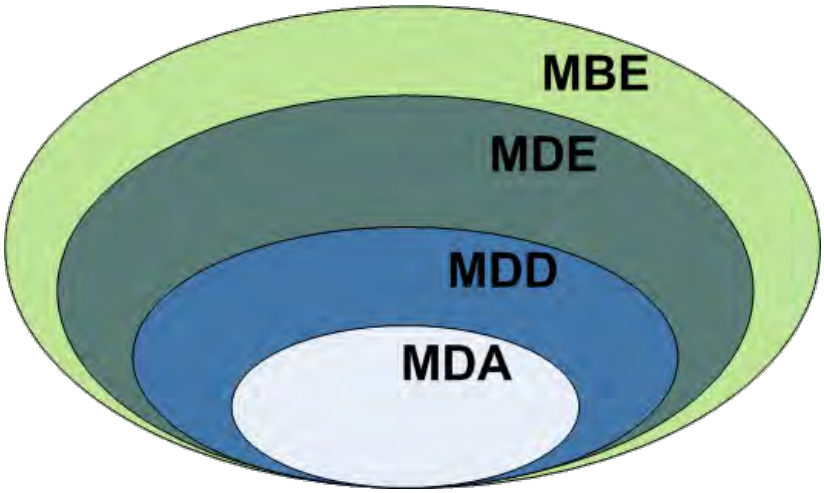


Figure II-1 : Représentation des initiatives dirigées par les modèles (Cabot, 2015)

II.2.2. Ingénierie Dirigée par les modèles (*Model Driven Engineering* – MDE)

Le MDE, terme proposé par (Kent, 2002), est une approche de développement qui propose d'élever les modèles au rang de concept de première classe (Bézivin, 2004). Il s'agit d'une forme d'ingénierie générative, qui se caractérise par une démarche rigoureuse par laquelle tout est généré à partir d'un modèle ce qui fait passer les modèles du statut de contemplatifs à celui de productifs. La différence entre le MBE et le MDE est que dans le premier les modèles jouent un rôle important mais ne sont pas nécessairement des artefacts clés du développement étant donné qu'ils ne dirigent pas le processus comme dans le MDE (Cabot, 2015). Par exemple, au lieu de générer le code directement à partir d'un modèle (comme c'est fait dans le MDE), dans le MBE, ce modèle est fourni au programmeur pour qu'il écrive le code manuellement.

Le MDE vise à augmenter le rendement des entreprises. Mais la question est de savoir si les entreprises sont prêtes à changer leurs habitudes et à adopter cette approche. Des études (Baker et al., 2005) (Whittle et al., 2014) (Burden et al., 2014) montrent que le résultat est assez prometteur. Si on prend le résultat des expériences de Motorola Europe, le degré de la maturité de modélisation a évolué de l'informel, sur tableau blanc (*whiteboard*), à la modélisation formelle. Ceci a conduit à une réduction approximative d'effort de 2,3 fois pour l'obtention du produit final. Cette réduction est due non seulement à l'utilisation de la génération automatique de code, mais aussi à l'utilisation de la simulation et des tests de modèles (*model testing*). Selon une autre étude (Hutchinson et al., 2014) menée sur des experts industriels en MDE, les modèles sont

utilisés pour plusieurs finalités. 95% des répondants utilisent les modèles pour la compréhension du problème, 91% pour la documentation et 62% pour la simulation des systèmes.

En plus de ces gains globaux, Motorola a observé d'énormes gains dans certaines phases du processus de développement. Par exemple une réduction de 30 à 70 fois a été observée dans le temps nécessaire pour corriger une anomalie détectée lors des tests d'intégration. Cette réduction est due à l'utilisation des tests de modèles qui permettent de résoudre le problème directement au niveau modèle plutôt que d'intervenir directement au niveau du code ce qui réduit considérablement le temps et ainsi le coût de la maintenance.

Pour résumer, les résultats recueillis au cours des dernières années ont montré les avantages du MDE par rapport à l'approche traditionnelle de développement en termes de qualité et de productivité :

- Qualité : une réduction globale de 1,2 à 4 fois du nombre des anomalies et une amélioration des anomalies de 3 fois en phase de maintenance d'anomalies. Le coût global de la qualité a également baissé en raison d'une diminution des temps d'inspection et de test,
- Productivité : une amélioration de la productivité de 2 à 8 fois en terme de lignes de code source.

L'approche MDE paraît très séduisante. Toutefois, il ne faut pas s'y méprendre, si elle a ses fervents adeptes, elle a aussi ses sceptiques voire ses détracteurs. Parmi les problèmes décelés par Motorola (Baker et al., 2005) on peut citer par exemple :

- Les mauvaises performances des outils lors du passage à l'échelle,
- L'absence d'outil supportant un environnement MDE complet,
- La difficulté à développer le système final en utilisant des environnements de développement hétérogènes,
- La présence de modèles isolés qui caractérise la mise en œuvre du MDE : même dans le cas d'un système fortement couplé, les modèles existent de façon séparée.

Ce sont ces inconvénients que notre approche traite dans cette thèse.

II.2.3. Architecture Dirigée par les Modèles (*Model Driven Architecture* – MDA)

L'approche MDA est une démarche proposée par l'OMG (OMG, 1989) depuis 2001. Il s'agit d'une vision particulière du Développement Dirigé par les Modèles (MDD: Model Driven Development) (Hailpern et Tarr, 2006). Ce dernier, qui contrairement au MDA, n'applique pas les standards de l'OMG, est un paradigme flexible pour la définition des processus de développement qui considère les modèles ainsi que les transformations comme des artefacts principaux de ce processus. Selon (Mellor et al., 2003) il s'agit tout simplement de la notion selon laquelle il est possible de construire le modèle d'un système afin de pouvoir par la suite le transformer automatiquement ou semi automatiquement en une chose réelle. Les artefacts du MDD sont utilisés pour spécifier, simuler, vérifier, tester et générer le système final.

À la différence du MDD, le MDE va au-delà des activités de développement et englobe d'autres tâches basées sur un processus d'ingéniérie logicielle (par exemple, l'évolution basée sur un modèle) (Cabot, 2015).

L'idée fondamentale du MDA, en utilisant les standards de l'OMG, est que les fonctionnalités du système à développer sont définies initialement dans un modèle indépendant des calculs (CIM : Computation Independent Model) qui est utilisé pour la création d'un modèle indépendant de toute plate-forme (PIM : Platform Independent Model). Ce dernier, épaulé par un modèle de description de la plate-forme d'exécution (PDM : Platform Description Model), permet la génération (semi-)automatique par transformation d'un ou d'un ensemble de modèles spécifiques aux plates-formes (PSM : Platform Specific Model). Les rôles de chacun de ces modèles sont les suivants :

- CIM : modèle indépendant de tout système informatique qui utilise un vocabulaire familier au maitre d'ouvrage. Il permet d'avoir une vision de ce qui est attendu du système, sans rentrer dans le détail de sa structure, ni de son implémentation. L'indépendance technique de ce modèle lui permet de garder tout son intérêt au cours du temps. Il est modifié uniquement si les connaissances ou les besoins métier changent,
- PIM : modèle qui décrit la logique métier ainsi que le fonctionnement des entités et des services. C'est un modèle qui ne contient pas d'information sur les technologies qui seront utilisées pour déployer l'application,
- PDM : modèle qui permet de décrire l'architecture logicielle de la plateforme d'exécution. Il contient les informations pour la transformation des modèles vers une plate-forme spécifique. Les outils BluAge Forward (Bluage, 2015a) et AndroMDA (Bohlen, 2007) définissent ce modèle sous forme de cartouche de génération remplaçable en fonction de la plate-forme d'exécution. Cette cartouche, appelée BSPs par BluAge (*BLUAGE Shared Plug-ins*), est disponible pour les frameworks les plus utilisés comme Struts, Spring, Hibernante, .Net, Java, etc.,
- PSM : modèle dépendant de la plate-forme technique spécifiée par l'architecte. Il sert essentiellement de base à la génération de code exécutable vers la ou les plates-formes techniques cibles. Il existe plusieurs niveaux de PSM. Le premier est issu de la transformation d'un PIM tandis que les autres sont obtenus par transformations successives jusqu'au code dans un langage spécifique (JSF2, EJB3, Struts, etc.).

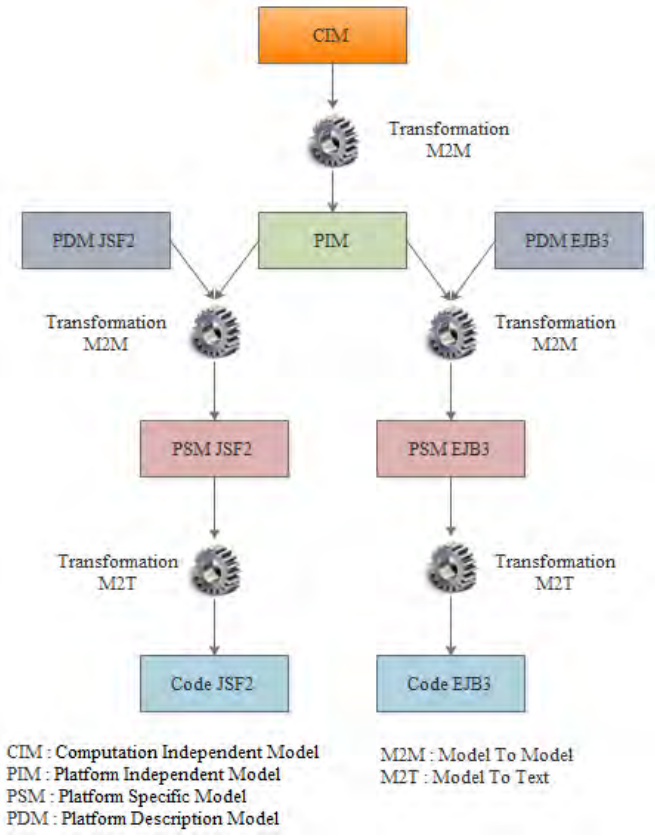


Figure II-2 : Exemple d'utilisation des modèles dans l'ingénierie vers l'avant (*Forward-engineering*)

Nous illustrons dans la Figure II-2 les différents types de modèles du MDA, utilisés pour la réalisation d'une application fictive. Pour aboutir à une application en JSF2 (Burns et al., 2009) par exemple, tout d'abord, le modèle CIM est transformé en modèle PIM. Ensuite, le modèle PSM est obtenu par une transformation qui prend en entrée le précédent modèle PIM ainsi que le modèle PDM qui décrit l'architecture logicielle de JSF2. Enfin le code de l'application est généré à partir du modèle PSM par une transformation M2T.

Dans la littérature de nombreux travaux ont traité le passage du PIM au PSM puis au code mais peu de travaux ont abordé le passage du CIM au PIM. Parmi ces travaux on peut citer (Bousetta et al., 2013) (Kherraf et al., 2008) (Kardoš et Drozdová, 2010) mais plusieurs questions concernant cet axe de recherche restent encore ouvertes.

Il faut aussi noter l'émergence d'une autre approche appelé ADM (*Architecture Driven Modernization*) qui est un effet miroir du MDA. Alors que le MDA se base sur une ingénierie vers l'avant (à partir du CIM), l'ADM se base sur la rétro-ingénierie. Elle consiste à extraire les artefacts du patrimoine logiciel à partir du code de l'application. BluaAge Reverse (Bluage, 2015b) est un exemple d'outil mettant en œuvre cette approche. Les artefacts extraits peuvent être utilisés avec une ingénierie vers l'avant afin de cibler une plate-forme technique comme c'est le cas de MyAppConverter (Greenfields-Development, 2015).

II.2.4. Métamodélisation et Multi-modélisation

Selon (Jézéquel et al., 2012), la métamodélisation est une activité consistant à définir le métamodèle d'un langage de modélisation. Elle vise donc à modéliser un langage qui joue le rôle du système à modéliser. Les notions de système, modèle et métamodèle ainsi que les relations entre elles sont présentées dans la Figure II-3.

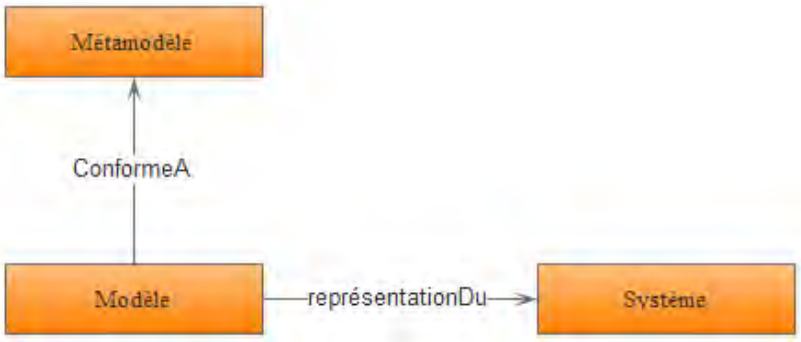


Figure II-3 : Relations entre système, modèle et métamodèle (Bézivin, 2004)

Comme évoqué dans la section II.2, un modèle est défini comme une représentation d'un système, construit pour un objectif précis. De cette définition découle la relation entre modèle et système nommée «représentationDu». Cependant, une fois le modèle construit, peut-on dire qu'il est une représentation correcte du système ?. Pour répondre à cette question le modèle doit pouvoir satisfaire le principe de substituabilité. Ce principe est défini par Minsky (Minsky, 1965) de la manière suivante : «un modèle doit être suffisant et nécessaire pour permettre de répondre à certaines questions à la place du système qu'il est censé représenter, exactement de la même façon que le système aurait répondu lui-même».

Etant donné que les modèles peuvent ne pas représenter l'ensemble du système et ainsi faillir à satisfaire le principe de substituabilité, plusieurs modèles peuvent être utilisés conjointement pour représenter le même système. Ceci est appelé «multi-modélisation». Selon (Bézivin, 2009) la multi-modélisation consiste à gérer un système complexe par l'intermédiaire de plusieurs modèles, chacun englobant un certain type de connaissance. Ces modèles nécessitent à un certain moment du processus de développement d'être composés pour obtenir un modèle plus global qui représente le système. Plusieurs approches s'intéressent à la multi-modélisation, dont les approches par points de vue, par aspects et par modèles.

Comme dit ci-dessus, un métamodèle est un modèle qui définit le langage d'expression d'un modèle, c'est-à-dire le langage de modélisation (Jézéquel et al., 2012). En d'autres termes un métamodèle est une représentation qui vise à définir les éléments utilisés dans un modèle. Le modèle est relié à son métamodèle par une relation nommée «conformeA». Il s'agit de la même relation qui relie un langage de programmation à sa grammaire.

De la citation «tout est modèle» de J. Bézivin (Bézivin, 2005), on déduit que le métamodèle est lui aussi un modèle et est donc conforme à un autre métamodèle (appelé méta-métamodèle). Pour assurer une cohérence des niveaux d'abstraction, l'OMG a défini une architecture de modélisation sur quatre niveaux (cf. Figure II-4) :

- Le niveau M0 correspond au monde réel. Il contient les informations réelles correspondant au système à modéliser. Il est conforme au modèle du niveau M1,
- Le niveau M1 (ou modèle) regroupe les modèles. Il décrit les informations de M0. Les modèles UML, PIM et PSM appartiennent à ce niveau. Le modèle M1 est conforme au métamodèle du niveau M2,
- Le niveau M2 (ou métamodèle), représente les langages de définition des modèles. Le métamodèle UML qui permet de définir des modèles UML, et le métamodèle SPEM qui permet de définir des modèles de procédé, appartiennent tous les deux à ce niveau. Il faut noter aussi que les profils UML, mécanismes d'extension du métamodèle UML, font partie de ce niveau. Un métamodèle est conforme à un méta-métamodèle,
- Le niveau M3 (ou méta-métamodèle) permet de décrire la structure des métamodèles et d'étendre ou de modifier les métamodèles existants. Le méta-métamodèle se décrit lui-même (réflexivité).

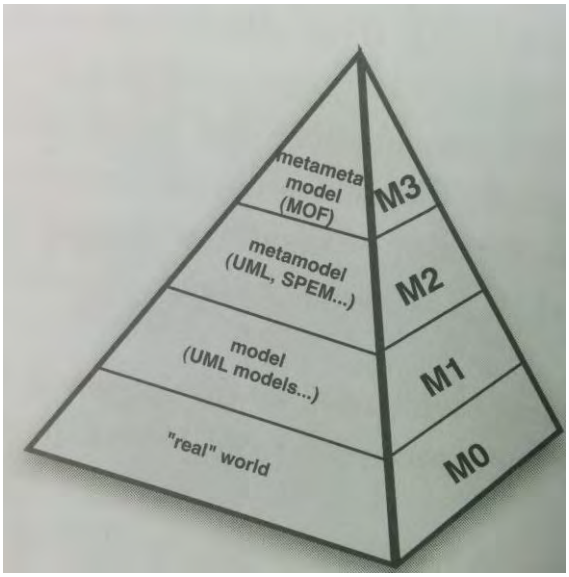


Figure II-4 : Pyramide de modélisation de l'OMG (Jézéquel et al., 2012)

Dans le cadre de cette thèse, nous nous intéressons uniquement aux niveaux M1 et M2, c'est-à-dire aux modèles et à leurs métamodèles.

II.2.5. Langages de modélisation

On distingue deux types de langages de modélisation : les langages de modélisation généralistes (*General Purpose Modeling Languages* – GPMLs) et les langages de modélisation dédiés (*Domain Specific Modeling Languages* – DSML).

Les DSMLs, ainsi que les DSLs (*Domain Specific Languages*), sont des langages dédiés chacun à un domaine métier spécifique et n'ont pas vocation à résoudre un problème en dehors de ce domaine. Le principal intérêt des DSMLs est qu'ils permettent aux experts d'un domaine métier de pouvoir penser en termes proches de leur domaine lorsqu'ils spécifient leurs systèmes (Bernoussi, 2008). Contrairement à un DSML, un GPML est un langage de modélisation

généraliste qui n'est pas restreint à un domaine particulier. Java et UML sont respectivement des exemples de GPL et GPML. Certes, les GPMLs sont (souvent) standardisés et sont supportés par de nombreux outils riches en termes de fonctionnalités, mais ils sont plus difficiles à apprendre et à utiliser. Pour UML, selon (Vallecillo, 2010), les utilisateurs ont de sérieux problèmes pour la compréhension de sa structure complexe et ont tendance à utiliser le peu qu'ils savent qui est estimé à 20%. Ce constat est consolidé par le résultat de l'étude de Hutchinson et al. (Hutchinson et al., 2014). La Figure II-5, décrit le nombre de diagrammes régulièrement utilisés par des experts industriels selon différents cas d'étude.

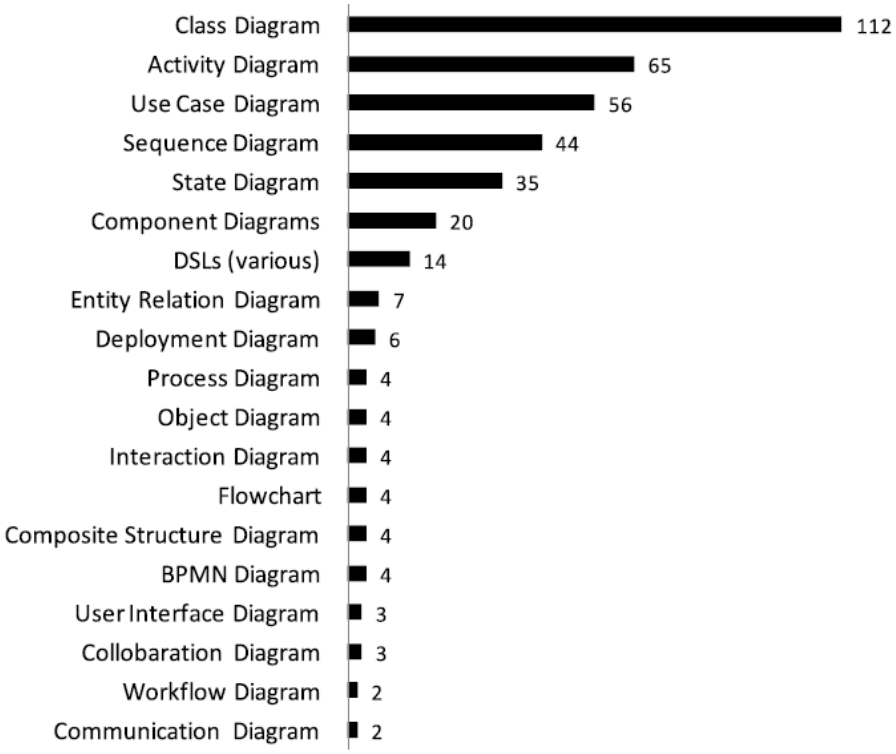


Figure II-5 : Liste des diagrammes utilisés régulièrement en UML (Hutchinson et al., 2014)

Un DSML est différent d'un GPML. Un GPML est un standard monolithique obtenu par consensus alors qu'un DSML offre aux utilisateurs des concepts propres à leurs métiers, ce qui permet de réduire le temps d'apprentissage du langage de modélisation. Un DSML permet aussi d'améliorer la productivité étant donné qu'il est plus facile de manipuler directement les concepts du domaine métier.

Un DSML est composé de trois éléments. Premièrement, la syntaxe abstraite qui décrit les concepts du langage et les relations entre eux. Deuxièmement, la (les) syntaxe(s) concrète(s) qui décri(ven)t le formalisme, textuel ou graphique, pour la manipulation du langage. Troisièmement, la sémantique qui attribue un sens à chaque concept du langage.

Martin Fowler (Fowler, 2010) a identifié trois types de DSMLs. Le DSML interne, le DSML externe et l'atelier de langage (*language worknbench*). Le DSML interne utilise un sous-ensemble du GPML mais dans un style personnalisé afin de gérer une partie de l'ensemble du système. G-Marker (Bluage, 2015a) est un exemple de ce type. Le DSML externe possède une existence autonome. Il correspond au type le plus utilisé et auquel on fait référence par abus de

langage par le terme DSML tout court. L'atelier de langage est un DSML pour la construction de DSMLs. Xtext (Bettini, 2013) et GMF (Gronback, 2009) sont des exemples de ce type.

L'utilisation de DSMLs a toutefois certains inconvénients, à savoir :

- Une cacophonie des langages due à l'utilisation de plusieurs DSMLs dans un projet,
- Les coûts de la conception, de la mise en œuvre et du développement des outils nécessaires risquent d'être élevés. Le détail sur les coûts de développment est présenté dans {White, 2009 #378} {Hoisl, 2013 #377},
- L'ajout progressif de nouvelles fonctionnalités peut faire évoluer le DSML en GPML.

II.2.6. Langages de transformation

La transformation est une opération fondamentale dans toute approche à base de modèles. Elle a pour objectif de rendre les modèles opérationnels ce qui augmente la productivité du développement. Selon (Kleppe et al., 2003), c'est une opération qui décrit comment une ou plusieurs constructions du langage source peuvent être transformées en une ou plusieurs constructions du langage cible. J. Bezivin (Bézivin, 2004) modélise la transformation par un modèle de transformation conforme à un métamodèle qui représente le langage de transformation (cf. Figure II-6).

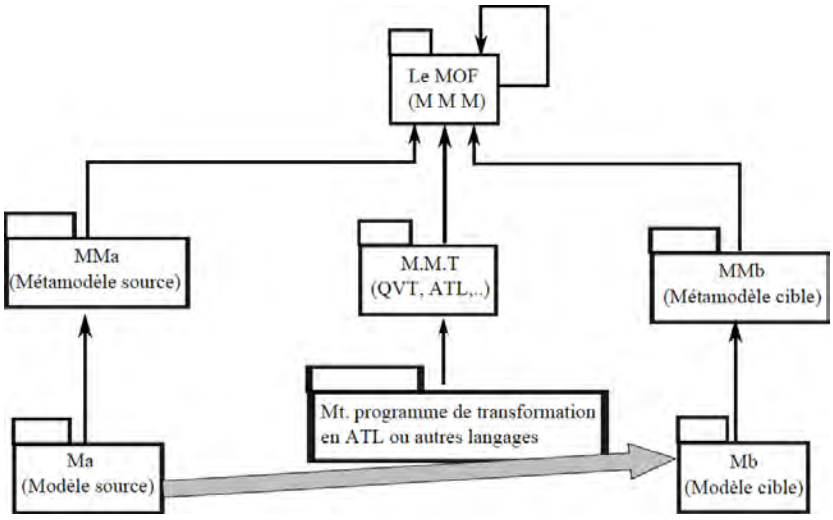


Figure II-6 : Principe d'une transformation de modèle (Bézivin, 2004)

Nous distinguons trois types de transformation : la transformation d'un modèle vers un autre modèle (Model To Model – M2M), la transformation d'un modèle vers un texte (Model To Text – M2T) et la transformation d'un texte vers un modèle (Text To Model – T2M). Parmi les langages de transformation M2M on peut citer : QVT (OMG, 2015), ATL (Jouault et al., 2008), VIATRA2 (Varró et Balogh, 2007), Xtend (Bettini, 2013), etc. Par exemple, le travail de (Pérez-Martínez et Sierra-Alonso, 2006) décrit la transformation d'un modèle d'analyse vers un modèle d'architecture logicielle. Dans les transformations M2M, Mens et al. (Mens et Van Gorp, 2006) distinguent une transformation «endogène» d'une transformation «exogène». Une tranformation

est endogène quand les métamodèles des modèles source et cible sont identiques alors qu'elle est exogène quand ils sont différents. Parmi les langages de transformations M2T, on peut citer JET (Eclipse, 2011b), Xpand (Klatt, 2007), Acceleo (Musset et al., 2006), etc. Le travail de (Albert et al., 2006) par exemple, décrit la transformation d'un modèle UML vers du code C#. Concernant la transformation T2M, elle est effectuée par une série d'étapes. Le texte est analysé en un modèle conforme au métamodèle de l'arbre syntaxique abstrait (Abstract Syntax Tree Metamodel - ASTM) (OMG, 2011a) puis transformé vers un modèle conforme au métamodèle de découverte de connaissances (Konwledge Discovery Metamodel - KDM) (OMG, 2011b). Ce dernier est transformé par la suite vers le modèle final. Le travail présenté dans (Barbier et al., 2011) décrit la transformation d'un code COBOL vers un modèle UML.

Dans la section suivante, nous présentons les travaux de notre équipe sur la modélisation par points de vue.

II.3. Profil VUML

La modélisation par points de vue constitue l'une des thématiques principales des travaux de notre équipe. C'est une approche de modélisation visant l'analyse et la conception des systèmes complexes avec une démarche centrée sur les acteurs interagissant avec le système.

Afin de modéliser un système par une approche combinant objets et points de vue, un profil UML appelé VUML (*View Based UML*) a été proposé (Nassar, 2003) .

Le principal ajout à UML est celui de «classe multivue». Une classe multivue est une entité de modélisation qui permet de décrire l'information en fonction des points de vue des acteurs concernés. Elle est composée d'une base partagée par les différents points de vue correspondant aux acteurs interagissant avec le système, et de vues qui étendent la base en décrivant les spécificités de chaque point de vue.

La Figure II-7 présente un aperçu du métamodèle UML étendu par le profil VUML. La sémantique informelle associée aux méta-éléments VUML (représentés en couleur grise) est la suivante :

- *Base* : méta-élément qui décrit les caractéristiques structurelles et comportementales communes aux acteurs du système,
- *View* : méta-élément permettant de modéliser les caractéristiques structurelles et comportementales spécifiques à un acteur donné,
- *GeneralView* : méta-élément représentant une vue abstraite,
- *ViewExtension* : méta-élément représentant une dépendance ayant comme source une vue et comme cible une base. Les vues dépendent de la base au sens où les attributs et les méthodes de la base sont implicitement partagés par les vues de la classe multivue,
- *ViewDependency* : méta-élément permettant la modélisation des relations de dépendance entre les vues. Chaque *ViewDependency* peut être associée à une ou plusieurs contraintes qui peuvent être exprimées soit en langage naturel, soit en langage formel tel qu'OCL,
- *MultiViewsClass* : méta-élément composé d'une *Base* et d'une liste de *View* reliés à la base via des *ViewExtension*.

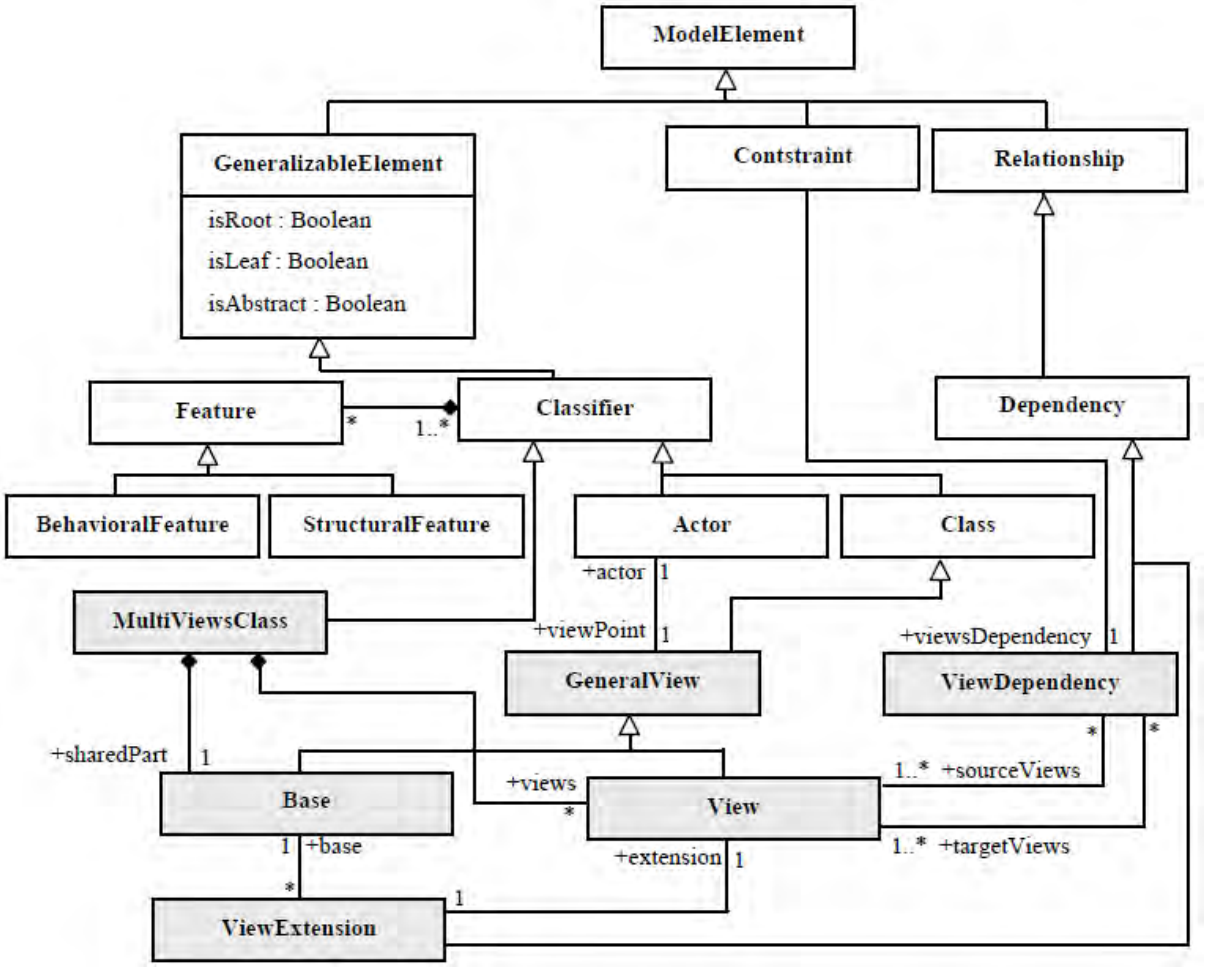


Figure II-7 : Fragment du métamodèle associé à VUML (Nassar, 2005)

Dans (Nassar, 2005) ces sémantiques ont été traduites en règles de bonne formation (*Well Formedness Rules*).

Le noyau de la démarche de conception VUML est le suivant. Premièrement, la modélisation des besoins est réalisée grâce aux cas d'utilisation. Deuxièmement, pour chaque point de vue, les scénarios ainsi que les diagrammes de classes associés (appelés aussi modèles partiels) sont spécifiés dans le formalisme UML. Troisièmement, un modèle VUML est élaboré par composition des modèles partiels. Afin d'automatiser le processus de composition, deux axes de recherche ont été explorés. Le premier axe concerne la composition des diagrammes de structures et plus particulièrement les diagrammes de classes (Anwar, 2009). Le second axe concerne la composition des diagrammes comportementaux et plus particulièrement les diagrammes d'états-transitions (Lakhrissi, 2010). L'automatisation de la composition dans les deux cas est fondée sur une transformation en langage ATL qui prend en entrée deux modèles partiels et produit en sortie un modèle conforme à VUML. La Figure II-8 illustre l'exemple d'un modèle VUML correspondant à la modélisation d'un système de gestion d'une agence de réparation de voitures. Les acteurs sont le chef d'agence, le responsable d'atelier, les mécaniciens, les clients, etc. Les principales classes multivue de ce système sont : Expertise, Reparation, Panne, ContratClient, Personnel et AgenceReparation. La base partagée (stéréotype «*base*») Voiture regroupe les attributs et les méthodes accessibles par tous les acteurs, tels que : *matricule*, *marque afficherInfosVoiture()*, etc. Par contre les vues (stéréotype «*view*») ne sont accessibles qu'à un acteur

spécifique. Par exemple, l'attribut *dateSortie* et la méthode *modifierInfosVoiture()* ne sont accessibles que par l'acteur ChefAgence.

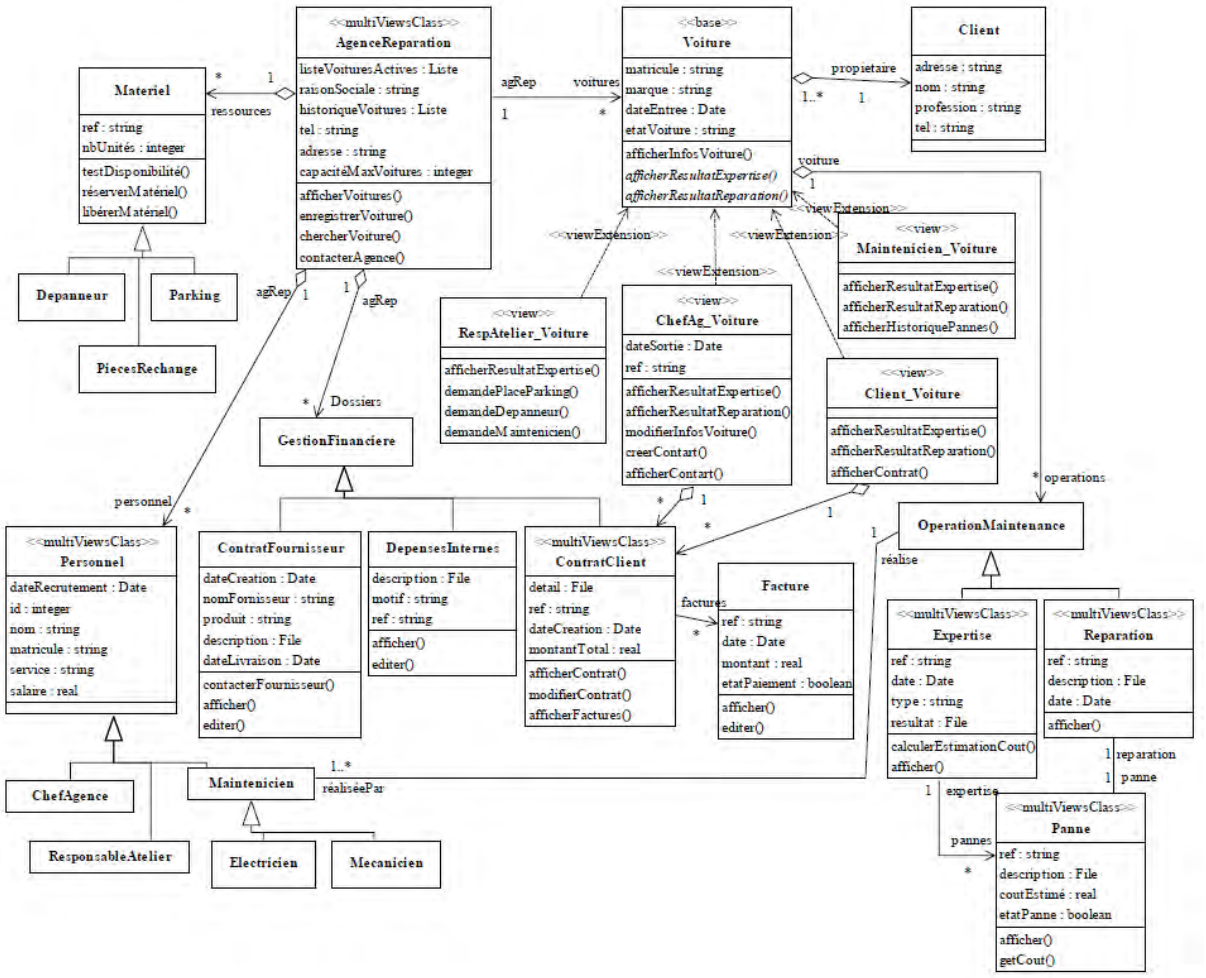


Figure II-8 : Exemple de modèle VUML (Lakhrissi, 2010)

II.4. Conclusion

Nous avons abordé dans ce chapitre les différentes initiatives dérivées du MBE (*Model Based Engineering*). Tout d'abord, nous avons mis l'accent sur le MDE (*Model Driven Engineering*) et son utilisation en industrie. Par la suite nous avons présenté le MDD (*Model Driven Development*) sur lequel se base le MDA (*Model Driven Architecture*). Puis, nous avons présenté les concepts de base liés à ces initiatives. Nous avons ensuite rappelé les travaux de recherche de notre équipe autour du profil VUML qui a pour objectif d'intégrer les concepts de vue et de point de vue dans l'analyse et la conception d'un système.

Le chapitre suivant présente l'état de l'art sur les trois thématiques que nous abordons plus spécifiquement dans ce travail de thèse, à savoir : la mise en correspondance, la composition et le traitement des évolutions de modèles.

CHAPITRE III. ETAT DE L'ART

"Get the habit of analysis. Analysis will, in time, enable synthesis to become your habit of mind"

— *Frank LloydWright*

III.1. Introduction

Nous nous intéressons dans ce chapitre aux travaux de mise en cohérence de modèles hétérogènes. La mise en cohérence consiste à produire une représentation du système et à maintenir son état compte tenu des éventuels changements sur les modèles. Dans la continuité de nos travaux sur VUML, nous avons réalisé un état de l'art sur la composition de modèles en deux phases. La première phase a consisté à étudier les approches de mise en correspondance en considérant cette dernière comme une opération élémentaire dans la définition de l'opérateur de composition alors que la seconde phase a traité de la composition proprement dite, consistant à produire le modèle composé à base d'informations récupérées de la première phase. Par contre, dès que les modèles évoluent dans le temps, le modèle composé n'est plus à l'image des modèles d'entrée. Une étude des approches de traitement de l'évolution de modèles s'est donc avérée nécessaire afin de pouvoir maintenir la cohérence entre le modèle composé et les modèles qui ont servi à sa création.

Ce chapitre est structuré en trois sections. Nous présentons dans la première section les approches de mise en correspondance. Nous passons en revue dans la deuxième section les approches de composition. Dans la troisième section, nous présentons les approches traitant l'évolution et la propagation des changements. Nous concluons ce chapitre par une synthèse globale mettant en évidence les limites des approches présentées.

III.2. Mise en correspondance de modèles hétérogènes

III.2.1. Introduction

En préambule, nous précisons que nous nous intéressons aux correspondances inter-modèles (et non intra-modèles) existant entre les différents modèles hétérogènes élaborés par les

acteurs des domaines métier d'un domaine d'application donné. Autrement dit, nous supposons que chaque modèle partiel à sa cohérence interne ou que celle-ci, abondamment traité dans la litérature, ne relève pas du champ de cette thèse. Par ailleurs, nous faisons une distinction entre une correspondance et un type de relation. L'égalité, la composition et l'héritage sont par exemple des types de relation ; une correspondance englobe le type de relation et les éléments reliés par celle-ci.

Le mécanisme de mise en correspondance est une des solutions utilisées pour traiter les problèmes de cohérence liés à l'hétérogénéité des sources de données. Il a pour objectif de définir la façon de relier les différents types d'artefacts.

Ce mécanisme joue un rôle central dans de nombreux domaines de l'informatique, tels que le Web sémantique (Fenza et al., 2008), les entrepôts de données (Banek et al., 2007), les systèmes embarqués (Eker et al., 2003), les bases de données (Castano et al., 2001), etc. Dans le domaine des bases de données par exemple, la mise en correspondance de schémas existe depuis longtemps.

De nombreuses approches de mise en correspondance ont été proposées durant la dernière décennie. Les schémas (en base de données avec le travail de (Spaccapietra et Parent, 1991)) et les ontologies (en web sémantique avec le travail de (Jiang, 1998)) en sont des initiateurs. Les travaux de Rahm et al . (Rahm et Bernstein, 2001) et Euzenat et al. (Shvaiko et Euzenat, 2013) synthétisent respectivement les approches phare dans ce domaine.

Dans le MDE, un consensus existe maintenant sur le concept de correspondance. Ceci a donné naissance à un nouveau domaine : la Mise en correspondance de (méta)modèles. Celle-ci est définie dans (Bézivin et al., 2006) comme une opération qui prend en paramètres un ensemble de modèles et produit en sortie un modèle de correspondance. Ce dernier contient les correspondances (liens) entre les éléments des différents modèles.

Dans ce qui suit, nous proposons un tour d'horizon des travaux dans ce domaine. Les recherches sur la mise en correspondance de modèles ont été initialement menées sur les diagrammes UML avec les travaux de Xing et al. (Xing et Stroulia, 2005) qui proposent un algorithme permettant leur comparaison. Avec l'émergence du MDE et l'évolution de l'utilisation des modèles, qui passe d'une exploitation générique (avec les GPMLs) à une exploitation spécifique (avec les DSMLs), de nouvelles approches ciblent des modèles spécifiques créés avec des langages dédiés. Les travaux décrits dans (Lin et al., 2007) sont parmi les premiers à traiter cette problématique.

Selon (Hausmann et Kent, 2003), l'opération de mise en correspondance doit satisfaire les exigences suivantes :

- Elle doit définir des correspondances entre différents modèles qui peuvent être hétérogènes. Autrement dit, la mise en place de l'opération de mise en correspondance ne doit pas être orientée vers un type de modèle partiel comme dans le cas des approches traitant uniquement les diagrammes UML.

- La représentation des correspondances doit être compréhensible pour permettre leur définition et leur approbation par les utilisateurs. Ces correspondances doivent permettre une présentation claire et concise des différents types de relation entre modèles,

• Les correspondances établies ne doivent pas interférer dans la définition des modèles d'entrée ; une séparation claire entre les modèles et leur(s) modèle(s) de correspondance est souhaitable,

• Les correspondances doivent être persistantes pour permettre de tracer les changements.

Nous ajoutons à cela l'exigence que les correspondances doivent être exprimées avec une sémantique précise. Ceci permet par exemple d'effectuer le traitement de l'évolution des modèles en entrée.

Dans la section suivante, nous étudions un processus de mise en correspondance générique. Nous examinons par la suite les approches existantes vis-à-vis des phases de ce processus qu'elles mettent en œuvre. Finalement, nous évaluons ces approches selon un ensemble de critères de comparaison que nous avons soit identifiés dans la littérature, soit déterminés comme étant pertinents dans le domaine de la multi-modélisation.

III.2.2. Processus de mise en correspondance

En examinant les approches de mise en correspondance abordées dans la littérature, nous constatons que l'opération de mise en correspondance est complexe et nécessite des notations spécifiques. Brun et al. (Brun et Pierantonio, 2008) ont proposé un processus générique qu'on peut appliquer à toutes les approches. Le processus consiste à décomposer l'opération de mise en correspondance en trois phases successives : Calcul, Représentation et Visualisation. Les deux premières phases sont importantes, compte tenu des exigences de l'opération de mise en correspondance définies précédemment.

III.2.2.1. Phase de calcul

Connue aussi sous le nom de «phase de comparaison de modèles», cette phase fait référence aux méthodes et techniques qui ont pour objectif d'identifier les différentes correspondances entre les modèles d'entrée. Elles varient entre celles qui s'appliquent sur des modèles spécifiques, telle que (Xing et Stroulia, 2005) et celles qui se basent sur une approche plus générique en prenant en entrée des (méta)modèles arbitraires, telles que (Van Den Brand et al., 2010, Lin et al., 2007) mais qui sont restrictives sur les types de relation considérés dans les correspondances.

Dans ce qui suit, nous décrivons différents mécanismes d'établissement de correspondances (section III.2.2.1.1) mis en place par une ou plusieurs techniques de calcul (section III.2.2.1.2).

III.2.2.1.1. *Mécanisme de mise en correspondance*

Au vu des approches étudiées nous distinguons quatre mécanismes de mise en correspondance, selon qu'ils sont :

- à base d'opérateurs, exemple : EMFCompare (Brun et Pierantonio, 2008),

- à base de patrons, exemple : Kompose (Morin et al., 2008),
- à base de règles, exemple : ECL (Kolovos et al., 2006b),
- à base de modèles, exemple : AMW (Del Fabro et al., 2005).

Mise en correspondance à base d'opérateurs

Dans ce mécanisme, le résultat de la phase de calcul est obtenu par application d'un ensemble d'opérateurs de correspondance qui se basent sur des heuristiques prédéfinies telles que la similarité de nom, de type, de structure, etc.

Mise en correspondance à base de patrons

Les correspondances sont construites à base de patrons, au sens Génie Logiciel du terme. Les patrons définissent des structures de correspondance génériques telles que les signatures, les points de jonctions (définis dans le modèle d'aspect (Kiczales et al., 1997)).

Mise en correspondance à base de règles

Ce type de mise en correspondance utilise un langage spécifique où l'utilisateur peut spécifier sa technique de calcul via des règles exprimées sous forme de syntaxe concrète textuelle. Les règles intègrent la sémantique de comparaison permettant d'exprimer des correspondances avec des types de relation plus complexes.

Mise en correspondance à base de modèles

Ce type de mise en correspondance permet de guider l'utilisateur dans le processus d'établissement des correspondances. Il consiste à créer un modèle de correspondance en se basant sur un métamodèle qui contient la définition des types de relation censés figurer dans le modèle.

III.2.2.1.2. *Les techniques de calcul*

Chaque mécanisme de mise en correspondance peut utiliser une ou plusieurs techniques. Selon les travaux de [KDRPP09], les techniques de calcul utilisées par les différentes approches de mise en correspondance peuvent être classées en quatre catégories :

- techniques à base d'identificateur statique (SI : *Static Identity*), exemple : (Alanen et Porres, 2003),
- techniques à base de signature (SIG : *Signature*), exemple : (Reddy et al., 2005),
- techniques à base de similarité (SIM : *Similarity*), exemple : (Treude et al., 2007, Lin et al., 2007),
- techniques à base de langage spécifique personnalisé (CLS : *Custom language-Specific*), exemple : (Xing et Stroulia, 2005).

Techniques à base d'identificateur statique

Ce type de correspondance utilise des identificateurs uniques persistants (sortes de clés primaires) pour chaque élément de modèle créé appelé UUID (*Universally Unique Identifier*) ou GUID (*Globally Unique Identifier*). L'encodage textuel d'un modèle utilisant *XML Metadata Interchange* (XMI) (OMG, 2014) est un exemple de standard utilisant cette technique.

Cette technique est le plus souvent appliquée pour comparer deux modèles construits à partir d'un autre modèle (ancêtre) ou bien pour comparer la version actuelle d'un modèle avec une version précédente (se trouvant dans l'historique local, par exemple). Pour que deux éléments correspondent, ils doivent posséder le même UUID. Ceci ne nécessite donc aucun effort de configuration de la part de l'utilisateur. Par contre, si on identifie une correspondance entre deux éléments a et b (à travers leurs UUIDs) et si on supprime l'un d'eux et le recrée à nouveau, l'élément se voit attribuer un autre identificateur et la correspondance n'est plus valide. De ce fait, cette approche n'est pas adaptée aux modèles qui sont susceptibles de changer ni à ceux qui sont construits indépendamment les uns des autres. De plus cette stratégie ne peut pas être appliquée aux technologies de modélisation qui ne supportent pas la gestion des UUIDs.

Techniques à base de signature

Les approches figurant dans cette catégorie, reposent sur l'utilisation des signatures comme technique d'identification non statique d'éléments de modèles. Comme décrit dans (Fleurey et al., 2008), les signatures sont calculées à partir des propriétés des éléments de modèles, en se basant sur des fonctions de calcul de ces signatures. Cependant, il revient à l'utilisateur de définir ces fonctions. Les auteurs admettent que ce genre de correspondance n'est pas approprié à tous les types d'éléments de modèle.

Techniques à base de similarité

Ces approches utilisent des fonctions de calcul de similarité entre deux éléments de modèle comme pat exemple *Jaccard* ou *Levenstein Edit Distance*. Un panorama des autres fonctions de calcul de similarité est présenté dans (Cheatham et Hitzler, 2013). Le degré de similarité est le plus souvent évalué par une valeur entre 0 et 1. De ce fait, les approches à base de similarité nécessitent une partie de configuration durant laquelle le seuil, limite à fixer pour décider si deux éléments sont similaires, est défini. La difficulté avec ce type de correspondance est d'évaluer le seuil à choisir afin de produire les meilleurs résultats. Les approches implémentant ce type de correspondance ne prennent pas en considération la sémantique des langages de modélisation.

Techniques à base de langage spécifique

Les approches à base de langage spécifique présentent l'avantage d'exploiter des données syntaxiques aussi bien que sémantiques pour améliorer la qualité du résultat obtenu par la mise en correspondance.

Néanmoins, avec ce type de correspondance, l'algorithme de comparaison doit être défini entièrement de façon manuelle. Pour pallier ce problème, nous pensons qu'il est possible de construire ce langage spécifique en utilisant conjointement les approches impératives (exemple : EMFCompare) et les approches déclaratives (exemple : ECL) obtenant ainsi des résultats plus précis. L'utilisation des approches impératives a pour objectif d'automatiser la création de

certaines correspondances dans le but de réduire l'effort requis alors que les approches déclaratives vont permettre à l'utilisateur de se focaliser sur la partie logique de la comparaison en exploitant l'infrastructure d'outil existante.

III.2.2.2. Phase de représentation

Identifier les correspondances n'est en fait qu'une phase du processus de mise en correspondance. En effet, le résultat de la phase de calcul précédente doit être représenté sous une forme qui permette son exploitation pour une analyse ultérieure par d'autres opérations (traitement de l'évolution, composition, etc.). Les correspondances peuvent être capturées sous forme de représentation dynamique ou bien sous forme persistante dans un fichier XML par exemple).

Les travaux de (Cicchetti et al., 2007) et de (Van Den Brand et al., 2010), proposent des propriétés que la phase de représentation doit s'efforcer de respecter pour qu'elle soit dissociée des méthodes définies dans la phase de calcul et pour permettre d'exploiter le potentiel offert par les plates-formes de modélisation. Ainsi, le résultat de la phase de représentation doit être :

- À base de modèle : Le résultat de la phase de calcul doit être représenté sous forme d'un modèle, afin de permettre différentes sortes de manipulation,
- Minimaliste : Le modèle de correspondance ne doit contenir que les éléments impliqués dans le mécanisme de mise en correspondance,
- Transformative : Il doit être possible de transformer un (ou plusieurs) modèle(s) partiel(s) vers un autre modèle en se basant sur le modèle de correspondance. Exemples : Fusion, tissage, etc.
- Indépendante d'un métamodèle particulier : Le modèle de correspondance ne doit pas cibler un domaine précis : il doit être indépendant des métamodèles métiers des modèles mis en correspondance.

III.2.2.3. Phase de visualisation

En faisant référence à un DSL, le résultat de cette phase représente la syntaxe concrète du résultat de la phase précédente (qui représente la syntaxe abstraite). Les modèles de correspondance produits doivent être représentés pour le concepteur sous une forme intuitive et lisible par des notations graphiques ou textuelles. Étant donné que les concepteurs ont chacun une manière spécifique de visualiser leurs connaissances, certains travaux ont proposé la production de plusieurs notations visuelles pour la même syntaxe abstraite. On peut citer par exemple le travail de Wouters et al. (Wouters, 2013).

La phase de visualisation – certes importante – est considérée tout de même comme secondaire par rapport aux étapes précédentes.

III.2.3. Les approches de mise en correspondance de modèles

Dans cette section, nous proposons un survol des approches de mise en correspondance en séparant pour chacune d'elle le mécanisme de mise en correspondance et les techniques de calcul utilisée. Nous commençons par les approches appliquées aux GPMLs suivies de celles appliquées aux DSMLs. Etant donné qu'il est possible par une technique appelé *lifting* (Kappel et al., 2007) de transformer un modèle en ontologie, nous présentons aussi à la fin une approche représentative de la mise en correspondance avec des ontologies.

III.2.3.1. Approches à base de GUID (*Globally Unique Identifier*)

RSA (*Rational Software Architect*) est un produit IBM qui intègre la mise en correspondance de diagrammes UML. Le modèle de calcul utilisé pour identifier les correspondances est basé sur un identificateur statique (GUID) attribué à chaque élément de modèle (voir section III.2.2.1.2 ci-dessus). La Figure III-1 illustre un exemple de comparaison de deux modèles sous RSA. Similairement à ce dernier, TopCased (Farail et al., 2006) − projet open source pour les applications critiques et le développement système − utilise la même stratégie de mise en correspondance (GUID). La différence avec RSA est que les modèles d'entrée sont conformes à des langages exprimés en EMF.

Ohst et al. (Ohst et al., 2003) présentent une technique dans la même veine que les deux précédentes dans le sens où elle permet de comparer les modèles en exploitant les GUIDs. Avant d'appliquer le processus de mise en correspondance, les diagrammes UML sont transformés en arbres. Par la suite ces arbres sont parcourus afin de trouver les éléments ayant le même GUID tels que les propriétés et les relations.

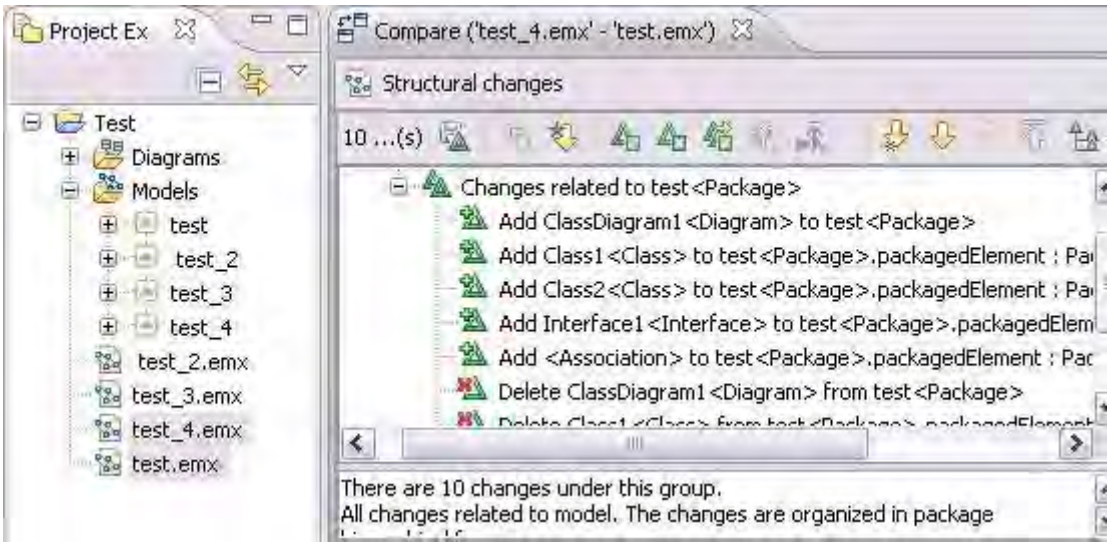


Figure III-1 : Exemple de mise en correspondance entre deux modèles test et test_4 (Letkeman, 2015)

III.2.3.2. UMLDiff / DSMDiff / SIDiff

UMLDiff (Xing et Stroulia, 2005) utilise des techniques de similarité (cf. III.2.2.1.2) à base de nom et de structure pour identifier les correspondances entre modèles UML. Le seuil des similarités est défini par l'utilisateur. Une extension de ce travail est décrite dans DSMDiff (Lin et al., 2007). Elle a l'avantage de supporter différents DSMLs spécifiés à l'aide de l'environnement générique de modélisation GME (*Generic Modeling Environment*). L'identification des correspondances est effectuée en deux étapes. La première étape est fondée sur une mise en correspondance à base de signatures, tandis que la seconde est fondée sur la similarité structurelle afin d'affiner le résultat de la première.

L'approche SIDiff (Treude et al., 2007) permet de comparer les modèles en se basant sur la similarité à condition qu'ils soient représentés au format XMI. Le processus de mise en correspondance commence par transformer les modèles XMI d'entrée en une représentation interne à base de graphes. Par la suite les correspondances sont identifiées par un parcours du graphe de bas en haut en appliquant des fonctions qui retournent des valeurs comprises entre 0 et 1. SiDiff fournit également le moyen de spécifier des algorithmes personnalisés de calcul de similarité structurelle.

III.2.3.3. SAMT4MDE

SAMT4MDE (Lopes et al., 2006a), extension du Framework MT4MDE (Lopes et al., 2006b), permet de générer des règles de transformation (en ATL) à partir d'un processus de mise en correspondance. L'établissement des correspondances s'effectue au niveau métamodèle. Il est fondé sur un algorithme qui consiste à retrouver les paires d'éléments qui sont en relation d'équivalence ou de similarité structurelle en se basant sur des métriques de calcul de distance à savoir Levenshtein and N-Gram (Cheatham et Hitzler, 2013). Une valeur de seuil a été définie à 0.7 afin de considérer que deux éléments sont similaires. Si la valeur vaut 1 les éléments sont équivalents. Après la création du modèle de correspondance, un utilisateur intervient afin de valider ou refuser les correspondances produites.

III.2.3.4. L'approche de Falleri el al.

Falleri et al. (Falleri et al., 2008) proposent une approche de mise en correspondance inspirée de (Melnik et al., 2002). Le processus de mise en correspondance se déroule en 3 phases (cf. Figure III-2).

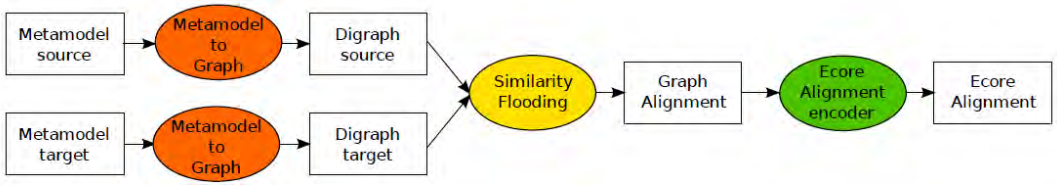


Figure III-2 : Vue générale de l'approche de (Falleri et al., 2008)

La première phase consiste à transformer les deux métamodèles d'entrée en graphes orientés étiquetés. Les auteurs proposent plusieurs stratégies pour transformer un métamodèle en graphe. Chacune de ces stratégies possède ses propres techniques de transformation. La deuxième phase permet d'appliquer l'algorithme de *Similarity Flooding* sur les graphes produits. Les graphes sont utilisés dans un processus de calcul de point fixe dont le résultat indique les nœuds dont les noms sont similaires dans les deux graphes. La troisième phase consiste à générer à partir du résultat de la phase précédente un modèle conforme au métamodèle de la Figure III-3.

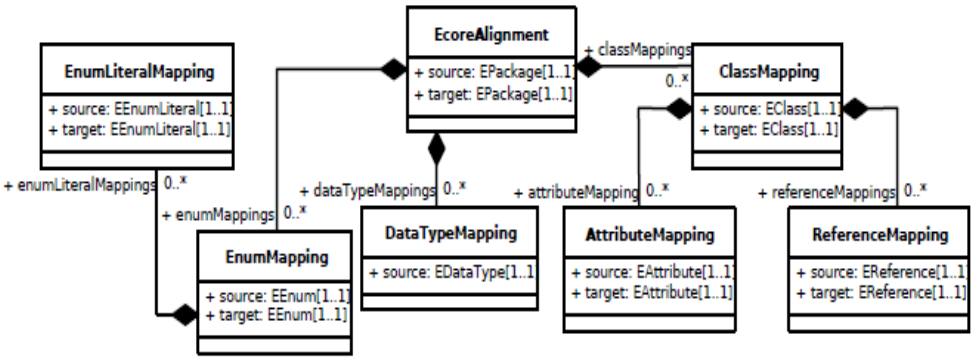


Figure III-3 : Métamodèle de correspondance (Falleri et al., 2008)

III.2.3.5. EMFCompare

EMFCompare (Brun et Pierantonio, 2008) est un Framework qui permet d'effectuer la mise en correspondance de modèles EMF. Cette approche permet de calculer les correspondances en se basant sur le principe de similarité. Le moteur de correspondance utilise des heuristiques et les éléments sont comparés suivant plusieurs métriques : similarité de nom, de type de contenu, de relation, etc. Elles retournent des valeurs variant de 0 à 1, qui sont regroupées afin de fournir des valeurs globales de similarité.

III.2.3.6. Epsilon Comparaison Language

ECL (Kolovos et al., 2010) est un langage d'établissement de correspondances à base de règles s'appuyant sur des techniques de calcul à base de langage spécifique personnalisé et à base de signature (voir section III.2.2.1.2). Il s'appuie sur la plateforme Epsilon sur laquelle il est possible de définir des langages de gestion de modèles, centrés sur des tâches spécifiques (*task-specific language*), comme la validation, la transformation, la génération, la comparaison et la fusion de modèles.

Une règle ECL prend en entrée deux paramètres faisant référence aux éléments de modèles à comparer. Le corps d'une règle ECL est défini en trois parties (Éléments surlignés de la Figure III-4) : une comparaison (*compare*), une mise en conformité (*conform*) et une troisième partie optionnelle de garde (*guard).*

La comparaison détermine si deux instances correspondent selon un ensemble de critères. La mise en conformité est un raffinement de la comparaison dans le sens où elle est exécutée seulement sur les éléments qui ont satisfait les critères définis dans la comparaison. Ainsi, par

exemple, on pourra vérifier si deux éléments de modèles possèdent le même nom. Si cette comparaison retourne une valeur vraie, on pourra vérifier la conformité en comparant des propriétés supplémentaires (type, cardinalité, etc.).

En ce qui concerne la partie optionnelle (*guard*), on peut y limiter l'application des règles de comparaison et de mise en conformité, à un sous-ensemble d'éléments définis. Ainsi les éléments qui ne sont pas déclarés dans cette partie ne seront pas affectés par l'exécution des règles.

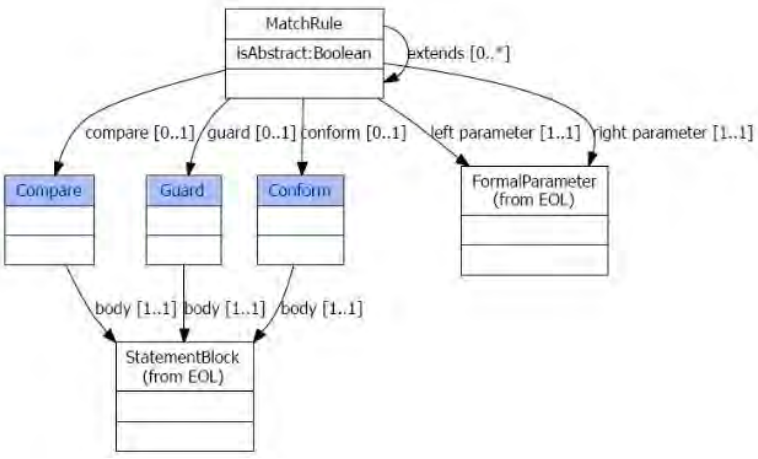


Figure III-4 : Métamodèle d'ECL (Kolovos et al., 2006b).

Le résultat de l'exécution des règles ECL fournit des paires d'instances, sauvegardées dans ce que les auteurs appellent «des traces de correspondance» (*matching trace*). Il s'agit d'un modèle conforme au métamodèle ci-dessous (Figure III-5).

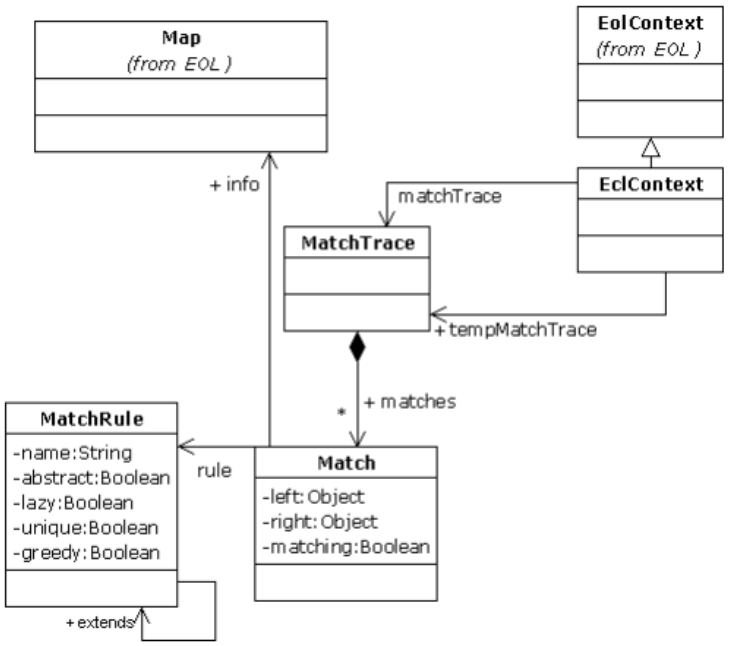


Figure III-5 : Métamodèle de trace de correspondances (Kolovos et al., 2010)

III.2.3.7. Atlas Model Weaver (AMW)

AMW (Del Fabro et al., 2005) est un Framework développé dans le cadre d'AMMA (*ATLAS Model Management Architecture*). AMMA est une plate-forme de gestion de modèles composée de quatre blocs fonctionnels : Transformation de modèles (*ATL: AtlanMod Transformation Language*), Tissage de modèles (*AMW: ATLAS Model Weaver*), Gestion de modèles globaux ou mega-modeling (*AM3: AtlanMod MegaModel Management*) et Projection de modèles (*ATP: AtlanMod Technical Projectors*).

AMW est applicable dans plusieurs buts comme l'interopérabilité, l'annotation de modèle, la composition, etc. Dans le cas de la composition, AMW est destiné au tissage de modèles. Nous présentons ici une pré-phase de cette composition consistant à établir les correspondances entre les modèles d'entrée. Ces dernières sont instanciées à partir d'un métamodèle appelé MMW (MetaModel Weaver). Elles sont par la suite sauvegardées dans un modèle de tissage appelé WM (Model Weaver).

Le MMW ne contient à l'état initial que les concepts de base. L'extension de ce métamodèle est obligatoire pour représenter les types de relation (exemple : similarité, égalité, etc.) nécessaires à l'établissement des correspondances entre les «modèles tissés» (*Woven models*). Cette extension doit être faite manuellement étant donné que les décisions sur les types de relation à représenter dans les correspondances sont fondées sur des connaissances humaines. La Figure III-6 fournit une description du métamodèle AMW. C'est le concept *WLink* qu'il faut étendre pour alimenter le MMW avec les types de relations nécessaires.

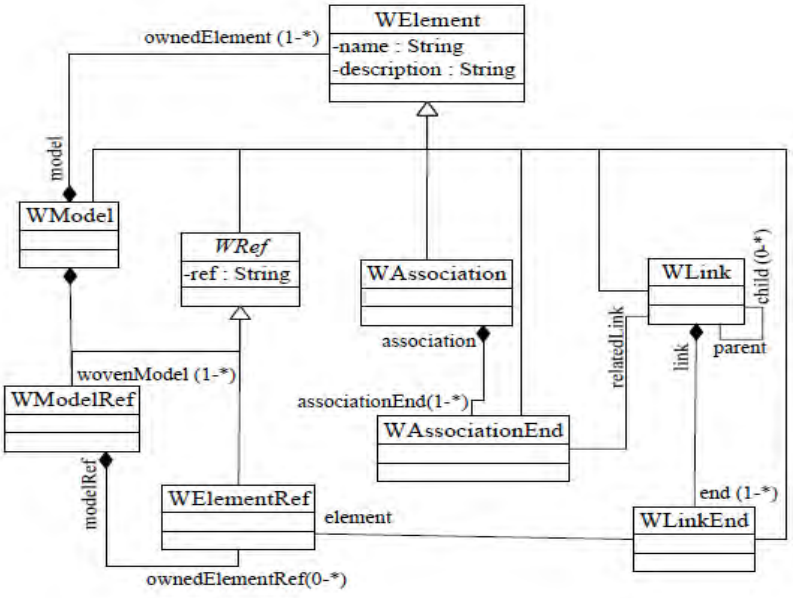


Figure III-6: Métamodèle AMW [DFBJ+05]

Le processus de mise en correspondance s'effectue en deux étapes. La première étape consiste à définir le métamodèle de correspondance (car celui proposé par AMW n'est qu'un noyau de base).

La deuxième étape a pour objectif la création manuelle d'un modèle de tissage contenant les correspondances entre les différents éléments des modèles d'entrée. Ceci est défini dans (Del Fabro et Valduriez, 2007) par l'opération suivante :

MW : MMW = Match (Ma : MMa, Mb : MMb).

Il est possible de générer le modèle de tissage de façon semi-automatique en utilisant des librairies de calcul de similarité. Ces librairies utilisent par exemple des techniques de similarité de nom, de similarité structurelle, de SF (*Similarity Flooding*) (Melnik et al., 2002), etc.

III.2.3.8. Kompose

Kompose est un Framework implémenté en Kermeta (Drey et al., 2009a) destiné à la composition de métamodèles. Il propose une approche conçue pour la modélisation orientée aspects. Étant donné que ce sont les métamodèles qui sont composés, les préoccupations transverses sont définies dans un métamodèle d'aspect alors que les fonctionnalités de base le sont dans un métamodèle primaire. Le processus de mise en correspondance est implicite, il est assuré à travers des techniques de calcul de correspondances à base de similarité de noms et de signatures. Une signature est un ensemble de propriétés syntaxiques associées à un type d'élément (Acher et al., 2010). La signature d'un élément de modèle comprend les valeurs liées à ses propriétés.

Le processus de mise correspondance est automatique, mais après la spécialisation de certaines opérations de comparaison. La Figure III-7 montre un exemple de définition de l'opération d'égalité «*equals*» qui est ajoutée par la suite dans le méta-métamodèle Ecore (cf. Figure III-8).

```
require kermeta
using kermeta::standard
using kermeta::kunit
class A {

 reference x : Integer

 method equals(compared : Object) : Boolean is do
  var castedCompared : A
  castedCompared ?= compared
  result := x.equals (castedCompared.x)
 end

}
```

Figure III-7: Exemple de définition d'une opération avec Kermeta (Drey et al.)

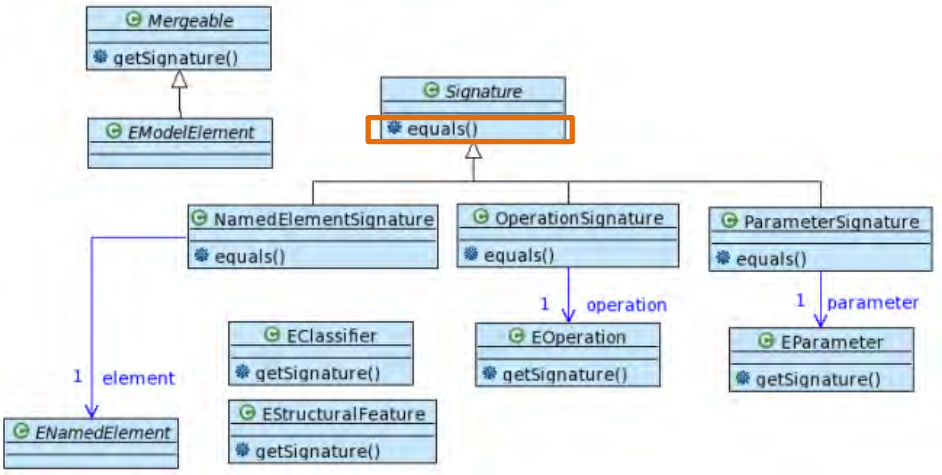


Figure III-8 : Ajout de l'opération d'égalité au niveau méta-métamodèle (Fleurey et al., 2008)

III.2.3.9. MatchBox

MatchBox est un Framework qui permet de combiner plusieurs stratégies se basant sur la technique de correspondance à base de similarité. L'utilisation conjointe de plusieurs stratégies permet de fournir des correspondances plus précises comparées à l'utilisation d'une seule stratégie.

Le processus de mise en correspondance (cf. Figure III-9) consiste en 3 étapes séquentielles (Voigt et al., 2010) :

- Étape 1 : Transformation des métamodèles d'entrée vers le modèle de données AMC (*Auto Mapping Core*), un modèle interne d'une structure arborescente dont l'implémentation est inspirée de COMA++ (Do, 2007). La raison de cette transformation est de pouvoir bénéficier des différentes approches de mise en correspondance existantes et de les exploiter sur le modèle AMC,
- Étape 2 : Le système MatchBox permet d'appliquer plusieurs algorithmes de mise en correspondance (appelés *Matchers*) à base de similarité. Il peut être configuré afin de permettre au développeur de choisir le type d'algorithme à utiliser et de produire pour chacun d'eux une matrice qui contient des valeurs de similarité entre tous les éléments du modèle. Les *matchers* implémentés sont : *name matcher*, *name path matcher*, *parent matcher*, *children matcher*, *sibling matcher*, *leaf matcher* et *datatype matcher*. Les différentes matrices obtenues sont regroupées sous forme de cube (Les coordonnées X, Y et Z représentent respectivement les éléments source, cible et le type de relation de la correspondance),
- Étape 3 : Cette étape a pour but de réduire le cube résultant de l'étape précédente à une matrice. Cette matrice est obtenue en définissant d'abord un seuil de similarité puis en filtrant les entrées du cube pour ne garder que les valeurs supérieures au seuil.

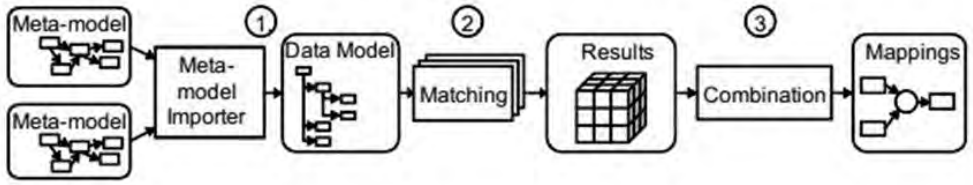


Figure III-9: Processus de MatchBox (Voigt et al., 2010)

Voight et al. (Voigt et Heinze, 2010) ont apporté des modifications à MatchBox, en intégrant la stratégie Planar GED. Cette approche apporte un changement au modèle interne des données qui a une représentation en graphe au lieu d'une représentation arboresente.

III.2.3.10. AML

AtlanMod Matching Language (Garcés et al., 2009) est une extension d'AMW permettant d'obtenir un modèle de correspondance. Ce DSL intègre différentes stratégies de mise en correspondance qui sont implémentées comme des séries de transformations de modèles. Chacune de ces transformations prend un ensemble de modèles en entrée et produit un modèle de correspondance en sortie (sous forme de modèle de tissage exprimé en AMW).

La définition d'un programme AML se fait en 3 étapes (Garces et al., 2010) :

- Importation des algorithmes AML existants pour les réutiliser,
- Déclaration des règles de correspondance,
- Définition du bloc de flux de modèle. Cette étape permet de préciser la façon dont plusieurs techniques de mise en correspondance interagissent afin de produire les différents liens de correspondance.

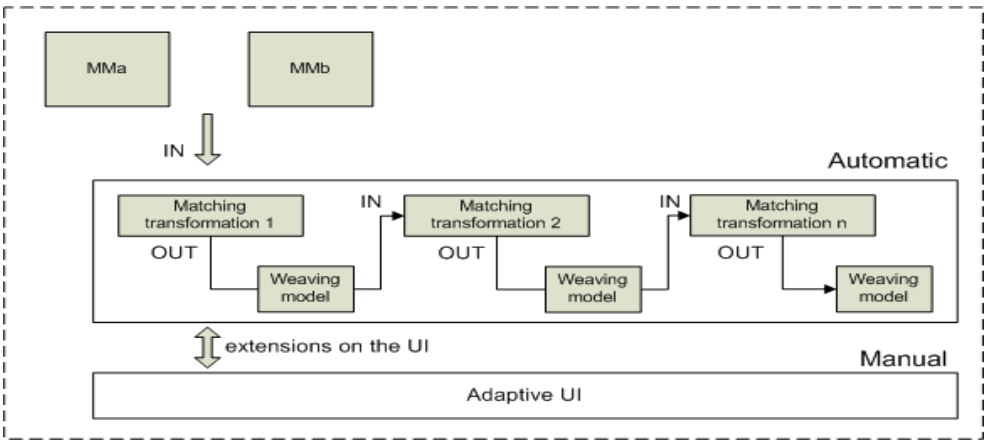


Figure III-10 : Processus AML

Comme le montre la Figure III-10, les transformations implémentent différentes heuristiques pour produire le modèle de correspondance. Au départ, la transformation prend deux métamodèles en entrée, et crée un modèle de tissage. Par la suite le mécanisme de

transformation utilise, en plus des entrées précédentes, le modèle de tissage produit lors de l'étape précédente afin d'affiner le modèle de tissage et de définir des correspondances plus précises respectant l'ensemble des heuristiques.

III.2.3.11. AgreementMaker

AgreementMaker (Cruz et al., 2009) est un Framework qui prend en entrée deux ontologies. Il combine un ensemble de techniques de mise en correspondance qui se basent sur une interpolation linéaire des valeurs de similarité. Le Framework inclut deux modules. Le premier module se charge du calcul de la similarité alors que le deuxième module sélectionne un sous-ensemble de correspondances qui satisfont un seuil prédéfini.

Le calcul de similarité est obtenu à l'issu de trois étapes. La première étape compare les concepts des ontologies en utilisant les BSMs (*Base Similarity Matchers*) suivi des PSMs (*Parametric String-based Matchers*). Un BSM regroupe une variété de méthodes syntaxiques et lexicales et se base sur l'utilisation de wordNet comme thésaurus. Pour le PSM (c.f. Figure III-11), l'utilisateur peut choisir entre un ensemble de techniques de comparaisons de chaînes de caractères, telles que Edit-distance et Jaro-Winkler (Dreßler et Ngomo, 2014), et définir le processus de normalisation à utiliser (par exemple : *stemming*, *stop-word removing*, ou *link stripping*).

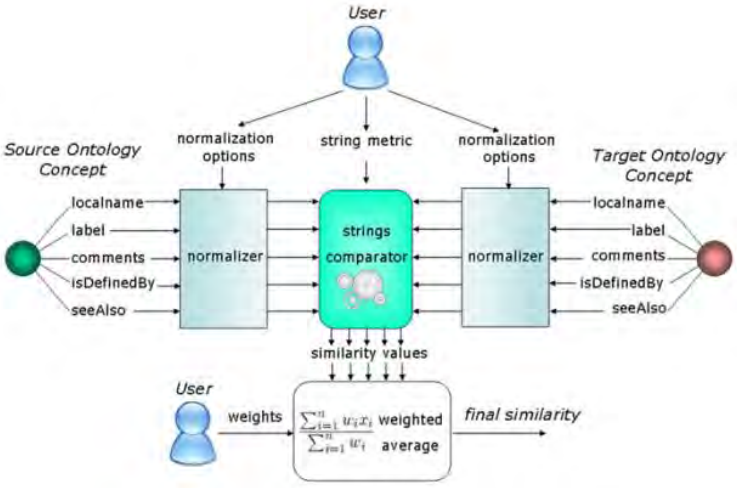


Figure III-11 : Partie paramétrage (PSM) du Framework AgreementMaker (Cruz et al., 2009)

La deuxième étape utilise les propriétés structurelles de l'ontologie. Elle utilise deux techniques qui sont la similarité des descendants (DSI: *Descendant's Similarity Inheritance*) (Cruz et Sunna, 2008) et la similarité des éléments du même niveau (SSC: *Sibling's Similarity Contribution*). Comme son nom l'indique, la première technique suppose que si deux concepts sont similaires, il y a une grande probabilité que leurs descendants le soient aussi. La même chose s'applique sur les concepts de même niveau pour la deuxième technique.

Finalement, dans la troisième étape, une combinaison linéaire pondérée (LWC: *Linear Weighted Combination*) est calculée sur la base des résultats provenant des deux niveaux précédents. Les résultats sont «purifiés» en fonction de seuils (*weights*) qui sont définis par l'utilisateur ou calculés automatiquement (c.f. Figure III-12).

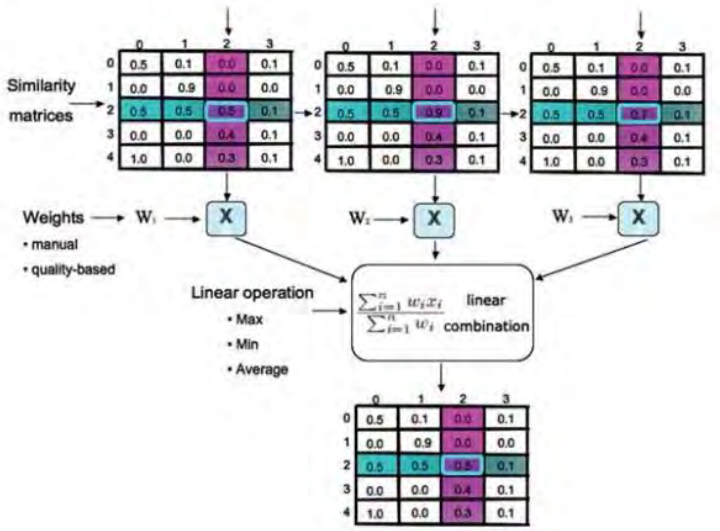


Figure III-12 : Combinaison linéaire pondérée dans AgreementMaker (Cruz et al., 2009)

III.2.4. Synthèse

Le but de cette synthèse est de définir dans un premier temps un ensemble de critères de comparaison pour les approches présentées plus haut étant donné que les validations empiriques ne sont pas toujours disponibles pour les différentes approches (les approches se basent sur un aspect théorique, avec des hypothèses et des intentions souvent implicites et non sur un aspect pragmatique avec une expérimentation industrielle). Dans un deuxième temps, une évaluation de la pertinence de chaque solution par rapport à ces critères est proposée.

III.2.4.1. Critères de comparaison

Afin de comparer les différentes approches présentées ci-dessus, nous avons utilisé des critères de comparaison définis dans (Do et al., 2003) à savoir : cardinalité, information auxiliaire utilisée, effort requis.

Pour permettre une caractérisation discriminante des approches et étant donné que nous nous intéressons à un environnement de multi-modalisation, nous avons ajouté un ensemble de critères que nous jugeons pertinents : hétérogénéité, nombre d'artefacts en entrée, type de représentation, stratégie de mise en correspondance, personnalisation et type d'artefact

Voici une brève description de chacun de ces critères :

- Cardinalité : Selon (Do et al., 2003) un élément d'un artefact peut participer à zéro, une ou plusieurs correspondances le reliant à un autre élément d'un autre artefact. On parle de cardinalité globale. Nous allons modifier légèrement cette définition afin de l'adapter au cas de la multi-modélisation. Une cardinalité est dite globale lorsqu'un élément d'un artefact peut participer à zéro, une ou plusieurs correspondances le reliant à d'autres éléments d'autres artefacts. En outre, une cardinalité locale évalue l'arité d'une correspondance établie uniquement entre les élements de deux modèles. Il s'agit de déterminer si un ou plusieurs éléments d'un

premier artefact peuvent être associés à un ou plusieurs éléments d'un second artefact dans une correspondance,

- Information auxiliaire utilisée : Des sources supplémentaires d'information telles que des dictionnaires ou des thésaurus peuvent être utilisées afin d'améliorer la qualité du résultat,
- Effort : Nous comparons aussi les approches en nous basant sur l'effort requis avant (pré-effort) et après (post-effort) la mise en œuvre du processus de mise en correspondance. Selon les auteurs (Do et al., 2003) ces critères permettent d'évaluer la qualité d'une approche. Le pré-effort inclut :
    - L'effort requis pour la compréhension du fonctionnement du processus,
    - l'effort requis pour la configuration des paramètres du processus de mise en correspondance, tels que la définition des valeurs des poids et le seuil à choisir.

  Le post-effort détermine le degré d'intervention humaine pour produire le résultat final. Il permet par exemple d'exprimer l'effort nécessaire pour supprimer les correspondances erronées et ajouter celles qui sont manquantes,
- Hétérogénéité : Ce critère indique que le processus de mise en correspondance peut prendre en entrée des modèles hétérogènes. Pour rappel, on considère que deux modèles spécifiques à un domaine sont hétérogènes si leurs langages de modélisation sont différents. Nous nous intéressons à des modèles issus de différents métiers, qui représentent la réalité de la multi-modélisation des systèmes complexes,
- Nombre d'artefacts en entrée : ce critère permet de caractériser la limitation éventuelle du nombre d'artefacts en entrée (le terme *artefact* est utilisé car nous ne comparons pas uniquement des approches se basant sur des modèles),
- Type de représentation : Ce critère décrit la forme de représentation d'un modèle de correspondance. Il s'agit de vérifier la possibilité d'exploiter le modèle de correspondance produit dans des opérations du MDE (composition par exemple),
- Technique de calcul : On considère ici les techniques traitées dans la section III.2.2.1.2 à savoir : SI (*Static Identity-Based*), SIG (*Signature-Based*), SIM (*Similarity-Based*) ou CLS (*Custom language-Specific*),
- Personnalisation : Le choix de ce critère est justifié par le besoin d'identifier les approches permettant d'exprimer des types de relation personnalisés selon le domaine d'application considéré.

III.2.4.2. Bilan des approches étudiées

Critères	Hétérogénéité	Nbr. d'artefacts en entrée	Cardinalité		Type de représentation	Technique de calcul	Personnalisation	Effort		Type d'artefact	I.A
Approches			générale	locale				Pré.	Post		
AgreementMaker	Oui	2	ND	m..n	Ontologie	SIM	Non	Définition du seuil+Choix des heuristiques à utiliser	Définition des cardinalités	Ontologie	Wordnet
AML	Oui	2	ND	1..n	Modèle	SIM	Non	Choix des heuristiques à utiliser, Ecriture des règles	Non	Métamodèle	Non
AMW	Oui	2	ND	1..n	Modèle	CLS	Oui	Définition du MMW, Choix des heuristiques à utiliser	Validation	Métamodèle	Non
DSMDiff	Oui	2	ND		Modèle	SIM, SIG	Non	Non	Non	Modèle	Non
ECL	Oui	2	ND	1..1	Modèle	CLS	Oui	Ecriture des règles	Non	Modèle	Non
EMFCompare	Oui	2	ND	1..1	Modèle	SIM	Non	Choix des heuristiques à utiliser	Non	Modèle	Non
Falleri et al.	Oui	2	ND	1..1	Modèle	SIM	Non	non	Non	Métamodèle	Non
Kompose	Non	2	ND	1..n	Modèle	SIM, SIG	Oui	Ecriture des règles	Non	modèle	Non
Matchbox	Oui	2	ND	1..1	Modèle	SIM	Non	Choix des heuristiques à utiliser	Non	Métamodèle	Non
RSA	Oui	2	ND	1..1	Modèle	SI	Non	Non	Non	Modèle	Non
SAMT4MDE	Oui	2	ND	1..1	Modèle	SIM	Non	Définition du seuil	Validation	Métamodèle	Non
SIDiff	Non	2	ND	1..1	Modèle	SIM	Non	Définition du seuil+ attribution des poids+personalisation de l'algorithme	Non	Graphe	Non
UMLDiff	Non	2	ND	1..1	Modèle	CLS	Non	Adapter l'algorithme	Non	Modèle	Non

ND : Non-défini
I.A : Information Auxiliaire

Tableau III-1 : Tableau récapitulatif des approches de mise en correspondance

Le Tableau III-1 résume l'évaluation des approches de mise en correspondance étudiées selon les critères identifiés dans la section précédente. L'étude de ces approches a permis de faire apparaître plusieurs limites.

Les auteurs d'AMW proposent un langage qui permet d'utiliser les transformations M2M pour comparer des modèles. Cependant, AMW n'est utilisable que lorsque les métamodèles source et cible sont similaires. Aussi, les concepteurs doivent étendre le métamodèle de manière à permettre la définition des types de relation, même pour les plus évidentes d'entre elles (exemple : similarité). Cette opération de mise en correspondance initialement faite manuellement, par l'intermédiaire d'un éditeur graphique, peut actuellement bénéficier d'un certain nombre d'heuristiques développées sous forme de librairies que le concepteur devra importer. Cependant, ces heuristiques sont uniquement à base de similarité structurelle.

Selon le même principe, EMFCompare permet de calculer les correspondances en se basant sur le principe de similarité. Le moteur de correspondance se fonde sur des heuristiques choisies par un utilisateur. Les éléments sont comparés suivant plusieurs métriques telles que la similarité de nom, de type de contenu et de relation. Dans le même contexte, SIDiff repose sur la similarité, mais un effort doit être fourni pour l'attribution des poids, la définition du seuil de similarité et la personnalisation de l'algorithme.

Dans SAMT4MDE, la mise en correspondance est réalisée en se basant uniquement sur la similarité. À la fin de la génération, un utilisateur intervient afin de valider les correspondances produites. Cette intervention peut être légère dans le cas d'un nombre peu élevé de correspondances à vérifier, mais risque d'être fastidieuse si le nombre de correspondances est élevé.

ECL est un langage qui permet la création d'un modèle de correspondance en exécutant une série de règles. Son utilisation est difficile car il requiert des compétences élevées et de gros efforts pour l'écriture des règles (les correspondances sont produites en exécutant une série de règles décrites en ECL). Il compare également les éléments de modèle un-à-un. En outre, pour exploiter ce résultat, le développeur doit ajouter des «sérialisateurs» afin de transformer les traces en un modèle de correspondance exploitable pour des opérations de type MDE, telle que la composition par exemple.

L'approche Kompose propose un processus de mise en correspondance qui doit être paramétré en définissant au niveau métamodèle des signatures spécifiques des opérateurs de mise en correspondance. À notre connaissance, cette approche ne s'applique qu'aux modèles homogènes.

L'approche de J.R. Falleri consiste à réaliser deux transformations. La première permet d'obtenir l'équivalent en graphes des métamodèles d'entrée. La seconde permet d'obtenir via le mécanisme de *SimilarityFlooding* uniquement les correspondances avec comme type de relation une similarité structurelle. De la même façon, MatchBox consiste à transformer, tout d'abord, les modèles d'entrée en un modèle arborescent : AMC (SAP Auto Mapping Core). Le processus se poursuit en appliquant un ensemble de stratégies afin de produire le modèle de correspondance. L'inconvénient majeur de MatchBox est que c'est au développeur de définir les transformations vers le modèle AMC.

Il existe plusieurs approches qui réalisent la mise en correspondance en se basant sur des ontologies. Nous avons choisi d'étudier AggrementMaker comme représentant des travaux à base d'ontologies, étant donné qu'il fournit les meilleurs résultats selon le travail de benchmarking de

Pavel Shvaiko et al. (Shvaiko et Euzenat, 2013), dans le contexte de l'alignement d'ontologies OAEI (*Ontology Alignement Evaluation Initiative*). L'avantage de l'utilisation des ontologies par rapport aux approches propres au domaine de la métamodélisation, est premièrement la puissance des méthodes syntaxiques et terminologiques pour la comparaison des chaînes de caractères et deuxièmement la possibilité d'avoir recours à des informations à partir de bases de connaissances telles que les dictionnaires, les thésaurus, etc. L'inconvénient de l'utilisation des ontologies est que les artefacts du domaine d'application doivent être définis conformément au domaine de l'ontologie. Autrement dit, une transformation vers le domaine des ontologies est nécessaire. En s'appuyant sur l'expressivité des ontologies, il est possible d'exprimer des correspondances complexes. Par exemple, un élément e1 est équivalent à l'intersection des éléments e2 et e3. L'inconvénient est qu'il n'est pas possible d'exprimer des types de relation autres que l'égalité et la similarité. Par exemple, il n'est pas possible d'exprimer qu'un élément correspond à l'agrégation de deux autres éléments. D'autres limitations telles que la restriction à un type de stratégie et la mise en correspondance au niveau TBox (équivalent à la notion de méta-élément) sont évoquées dans le paragraphe suivant.

D'autres inconvénients communs aux différentes approches peuvent être cités. Premièrement, en dehors du fait qu'il n'est pas prouvé que les approches qui s'appuient sur une transformation de modèle (ex : MatchBox, Falleri) ne ralentissent pas le processus de mise en correspondance, il n'est pas démontré qu'il n'existe pas une perte d'informations lors de cette transformation. Ce problème a été soulevé par les auteurs de MatchBox dans leur approche. Deuxièmement, les approches qui possèdent comme type d'artefact un (méta)modèle ne sont pas appropriées lorsque ces (méta)modèles ont une grande distance sémantique entre eux. Dans ce cas, selon (Kappel et al., 2007), la précision du résultat est très faible (généralement en dessous de 0,10). Le résultat est satisfaisant seulement quand il s'agit de métamodèles qui partagent les mêmes propriétés. Troisièmement, la mise en correspondance s'effectue entre méta-éléments (TBox en ontologie). Cependant ceci n'implique pas que leurs instances (Abox en ontologie) soient aussi en correspondance. Quatrièmement, le nombre d'artefacts en entrée est toujours de deux ce qui ne permet pas de créer des correspondances de cardinalité générale. Enfin, la majorité des approches visant l'automatisation, utilisent une mise en correspondance à base de similarité structurelle et n'exploitent pas d'autres types de relation tels que la similarité sémantique, la composition, la dépendance, etc.

Après avoir étudié les approches de mise en correspondance, nous étudions dans la section suivante les approches de composition de modèles.

III.3. Composition de modèles

III.3.1. Introduction

Aujourd'hui le développement logiciel repose sur un ensemble varié de langages, d'outils et de processus. Cette variété donne lieu à de multiples modèles souvent hétérogènes. Ajoutons à cela que les experts en modélisation sont souvent situés dans des espaces géographiques distribués. Afin d'obtenir une vision globale du système, une étape d'assemblage et d'intégration de ces modèles est indispensable dans le processus de développement.

Selon (Pillar, 2007), l'intégration est une activité qui combine des éléments d'un système en un système plus vaste. Elle possède une définition plus précise selon le domaine mis en œuvre. En technologie de l'information, l'intégration de systèmes est l'exploitation des données et de leurs dépendances dans un processus consistant à relier les différents composants des systèmes d'information pour agir comme un ensemble coordonné plus large ; cette définition fait référence à la mise en correspondance présentée dans la section précédente. Dans le domaine de l'ingénierie, l'intégration peut être définie comme un processus qui consiste à combiner différentes composantes en un seul système et à veiller à ce que chacune de ces composantes fonctionne au sein d'un tout fonctionnel. Cette définition est celle utilisée dans la composition de modèles. L'identification du problème de la composition en MDE a été initialement abordée par Oliveira et al. dans (Oliveira et De Oliveira, 2007), mais cette notion a été identifiée également dans plusieurs autres domaines : modélisation orientée aspects (Rausch et al., 2003), modélisation UML (Egyed et Medvidovic, 2000) (Anwar et al., 2010), bases de données (Buneman et al., 1992), web services (Liu et al., 2005), architecture orientée services (Mosser et al., 2010, Cavallaro et al., 2010), ingéniérie du logiciel basée sur les composants (Shashank et al., 2010), etc.

Dans (Bézivin et al., 2006) la composition est une opération qui consiste à produire un seul modèle par combinaison de deux modèles (par exemple MA et MB) en prenant en compte le modèle de correspondance (CAB). Si MAB est le modèle résultant, MAB = Compose (MA, MB, CAB). Plusieurs sujets de réflexion peuvent être identifiés concernant l'opération de composition comme le type de modèle source, la syntaxe et la sémantique du modèle composé, etc. Ces éléments sont abordés ultérieurement dans cette section.

Selon (Herrmann et al., 2007), la composition doit traiter trois niveaux :

- niveau syntaxique : Expression du modèle composé issu des modèles d'entrée,
- niveau sémantique : Définition d'une sémantique associée au modèle composé, en fonction de la sémantique des modèles d'entrée,
- niveau méthodique : Utilisation du modèle composé, issu du processus de composition, dans un processus de développement logiciel (exemple : intégration au processus de génération de code, d'évolution, etc.).

Avant de déclencher le processus de composition à proprement parler, il est nécessaire d'identifier les correspondances entre les éléments à composer. De ce fait, le processus de composition ne peut pas être considéré comme une opération atomique, d'où la pré-phase de mise en correspondance (section III.2). Cette pré-phase est suivie par une opération de composition proprement dite qui a pour objectif la création du modèle «global» en combinant les éléments des modèles d'entrée à l'aide des correspondances définies.

III.3.2. Les approches de composition de modèles

Dans cette section, nous présentons huit approches de composition de modèles, représentatives des différentes techniques mises en œuvre dans le domaine : UML2 Package Merge, Kompose, Epsilon Merging Language, Multi Document Integration, Atlas Model Weaver, Virtual EMF, l'approche de Wouters et al. et l'approche de fédération de modèles.

Pour présenter l'étude, nous proposons un processus générique (cf. Figure III-13) que nous avons appliqué sur les différentes approches afin de faciliter leur comparaison. Ce processus consiste tout d'abord à introduire les différents modèles et leurs métamodèles respectifs. Ensuite un modèle de correspondance est produit via une opération *Matching* qui prend en entrée les modèles d'entrée et produit en sortie un modèle de correspondance, conforme à un métamodèle de correspondance prédéfini dans chaque approche. L'étape finale est l'exécution de l'opération *Composition* qui produit un modèle global du domaine en combinant les modèles d'entrée, à l'aide des correspondances obtenues précédemment.

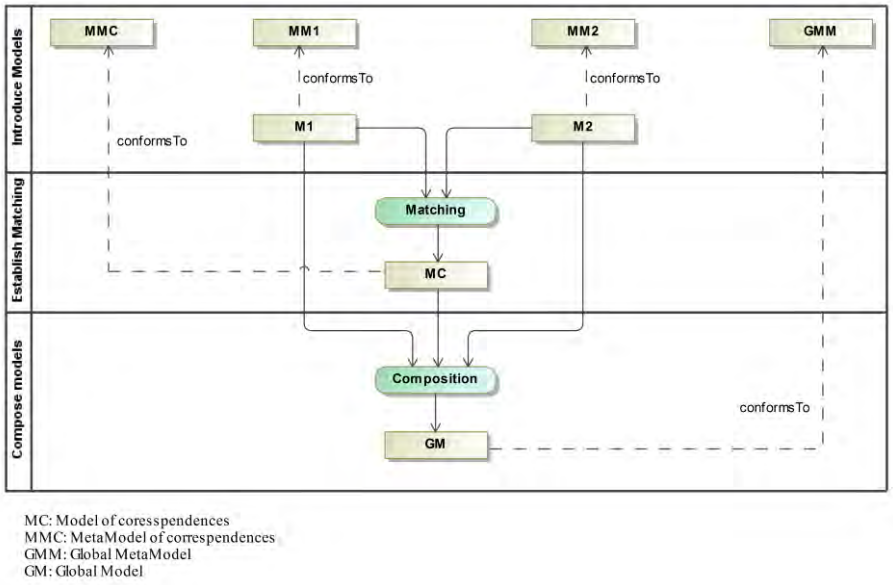


Figure III-13 : Processus générique de composition

III.3.2.1. UML2 package Merge (UML, 2011)

La composition en UML est un mécanisme de fusion du contenu de deux paquetages avec la notion de *Package Merge*.

UML 2 définit la notion de fusion de paquetages (OMG, 2011c) comme une relation dirigée entre deux paquetages : un paquetage récepteur (PR) et un paquetage fusionné (PM). Ce mécanisme de fusion intègre le contenu des deux paquetages dans le premier. Cette relation a été formellement définie par (Zito et al., 2006), avec l'expression suivante : PR := PM + PR.

L'objectif de la fusion de paquetages est l'extension des fonctionnalités. Autrement dit, la possibilité d'étendre les concepts définis dans un paquetage avec les fonctionnalités d'un autre (Dingel et al., 2008).

La fusion de paquetages implique un ensemble de contraintes et de transformations. Les contraintes sont des préconditions dont le rôle est de valider le processus de fusion. Par exemple, un paquetage ne peut pas être fusionné avec un autre s'ils sont reliés par une relation de composition ou d'agrégation. Les transformations sont considérées comme des post-conditions de la relation de fusion dont le rôle est la mise en correspondance des éléments puis la fusion des éléments correspondants, en corrigeant automatiquement, par des règles précises, certaines incohérences. Par exemple, si deux éléments possèdent la même visibilité alors l'élément résultant aura cette même visibilité, par contre si les deux visibilités sont différentes, la visibilité «publique» sera affectée par défaut à l'élément résultant.

La Figure III-14 présente le processus de composition d'UML2 Package Merge. M1 et M2 font respectivement référence à PR et PM. De ce fait GM n'a pas d'existence concrète. Ce n'est pas un nouveau modèle créé mais une copie de tous les éléments de M1 auxquels sont ajoutés les éléments de M2, modulo l'application des règles prédéfinies.

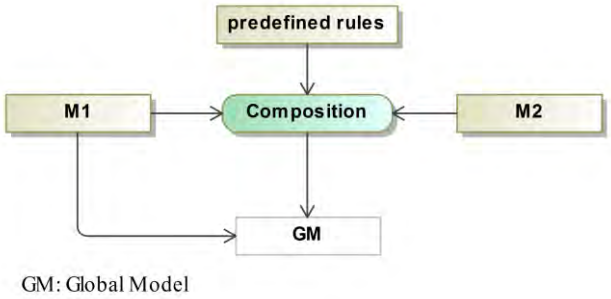


Figure III-14 : Processus de composition d'UML Package Merge

Le processus de composition est structuré en deux phases de mise en correspondance et de composition.

III.3.2.1.1. *Phase de mise en correspondance*

Cette approche ne propose pas une phase distincte de mise en correspondance dans laquelle un modèle de correspondance est produit. La mise en correspondance est mise en œuvre par des règles prédéfinies. IL s'agit d'une approche simpliste de comparaison à base de nom et de type des éléments de modèles. Ainsi par exemple deux éléments de modèles seront composés s'ils possèdent le même nom et le même type.

III.3.2.1.2. *Phase de composition*

Les éléments qui ont le même nom et le même type sont fusionnés alors que les autres sont copiés. Le modèle composé regroupe l'ensemble des propriétés des deux modèles sans appliquer de filtre sur le contenu du modèle composé.

Conceptuellement, le mécanisme de fusion de paquetage (PM) produit un autre paquetage en combinant le contenu des deux paquetages PM et PR, mais en réalité aucune distinction n'est faite entre le paquetage fusionné et le paquetage résultant de l'opération de la fusion.

III.3.2.2. Kompose

Kompose (Drey et al., 2009a) est un Framework qui implémente une approche de modélisation orientée aspects pour la composition de modèles à travers un ensemble de directives.

La Figure III-15 montre le processus de composition sous forme d'adaptation de notre processus générique.

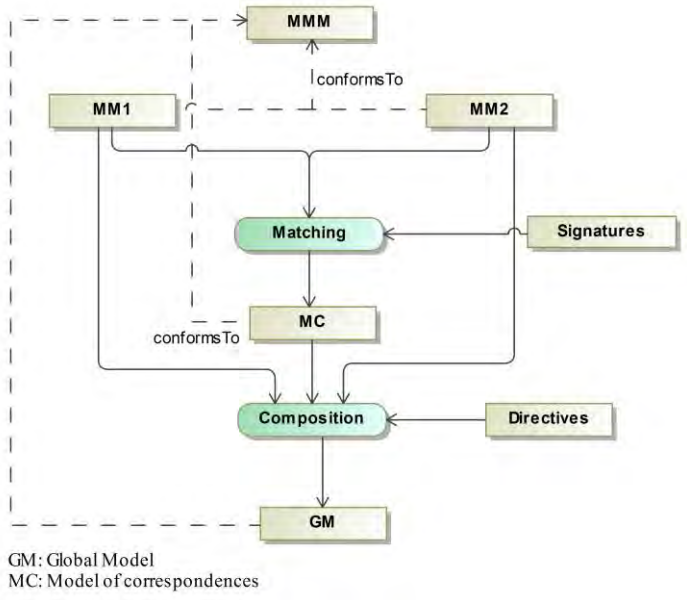


Figure III-15: Processus de composition de Kompose

III.3.2.2.1. *Phase de mise en correspondance*

La phase de mise en correspondance (*Matching* sur la Figure III-15) est fondée sur l'implémentation d'un ensemble d'opérations qui sont spécialisées en fonction du domaine d'application traité (voir section III.2.3.8).

III.3.2.2.2. *Phase de composition*

Cette phase est implémentée d'une manière générique en utilisant l'introspection (Morin et al., 2008). Les éléments ayant la même signature sont composés et leurs contenus (propriétés et méthodes) sont comparés à leur tour puis composés. Les éléments n'ayant pas de correspondant sont simplement copiés dans le modèle composé. Cette technique pose certains problèmes, dans le sens où elle ne permet pas de composer des éléments qui représentent le même concept lorsqu'ils possèdent des signatures différentes. C'est par exemple le cas de deux classes possédant le même nom, mais ayant des propriétés différentes (l'une abstraite et l'autre concrète).

Le mécanisme proposé pour résoudre ces conflits réside dans l'utilisation de directives de composition afin d'adapter les modèles source. Les directives de composition permettent de forcer des correspondances et d'en rejeter d'autres ou bien de contourner les règles de fusion par défaut en les redéfinissant (Fleurey et al., 2008).

Il existe deux types de directives :

- *pre-merge* : Directives permettant d'adapter les modèles d'entrée afin d'y introduire des modifications avant d'entamer la composition. Par exemple, dans le cas où deux éléments représentent le même concept avec deux noms différents, une directive de ce type est appliquée pour changer le nom de l'élément afin qu'il soit identique à l'autre élément,
- *post-merge* : Directives appliquées dans le but de corriger les incohérences ou bien les problèmes identifiés dans le modèle composé. C'est ainsi qu'on pourra par exemple éliminer des éléments de modèles qui ne doivent pas apparaitre pour une raison ou une autre dans le modèle composé. Par exemple, un aspect de sécurité peut exiger la suppression des associations présentes dans les autres aspects. Cette restriction est exprimée par une directive qui retire ces associations du modèle composé.

Le métamodèle des directives de Kompose (cf. Figure III-16) regroupe les directives utilisées dans les phases amont et aval de la composition (pré-directives et post-directives). Elles sont de deux types principaux : *Change* et *Create.*

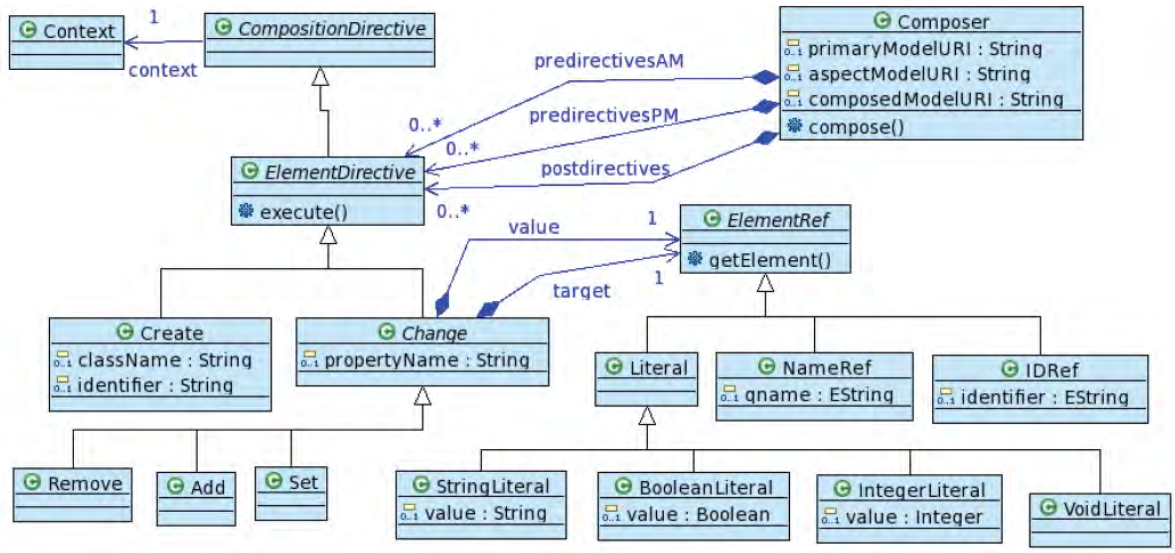


Figure III-16: Métamodèle des directives de Kompose (Fleurey et al., 2008)

L'approche MOMENT (Boronat et al., 2007) est un cas particulier de Kompose. Elle propose l'équivalence comme unique type de relation prédéfini en utilisant l'opérateur *equals*. Des stratégies de résolution de conflits sont possibles via l'opérateur *resolve*. À l'instar d'UML2 Package Merge, l'opérateur consiste à définir le modèle de référence. Lors d'un conflit, les éléments du modèle de référence sont privilégiés. Un troisième opérateur, *refresh*, est utilisé pour copier dans le modèle composé les éléments qui ne sont pas en équivalence.

III.3.2.3. Epsilon Merging language

EML (Kolovos et al., 2006a) permet la composition de modèles hétérogènes. C'est un langage à base de règles, fondé sur la plateforme Epsilon (*Extensible Plateform for Specification of*

Integrated Languages for mOdel maNagement). Epsilon est une plateforme de base : c'est un noyau sur lequel il est possible de définir des langages de gestion de modèles, centrés sur des tâches spécifiques (*task-specific language*) comme la validation, la transformation, la génération, la comparaison et la fusion de modèles.

La composition en EML est définie par trois types de règles. Comme le montre l'extrait de la syntaxe abstraite d'EML présenté sur la Figure III-17, ces types sont : comparaison, fusion et transformation.

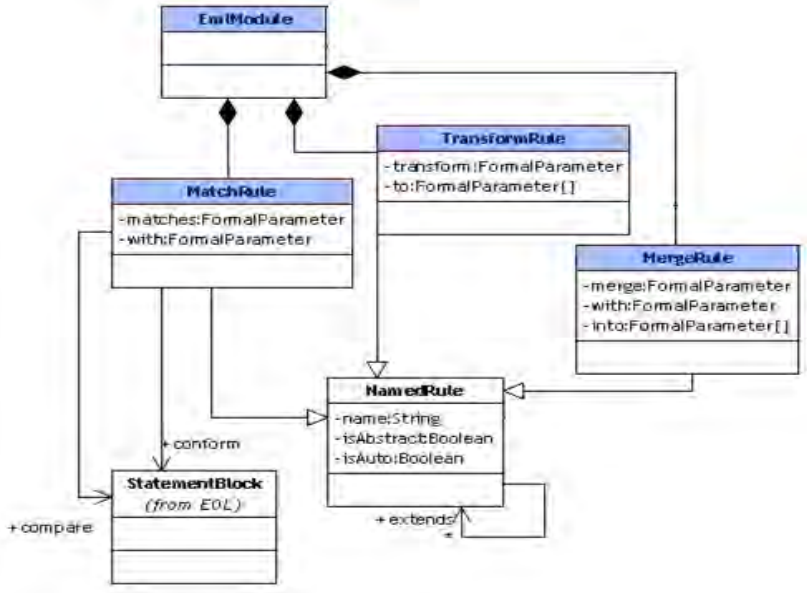


Figure III-17 : Syntaxe abstraite d'EML (Kolovos et al., 2006a)

Le schéma de la Figure III-18 présente le processus de composition d'EML.

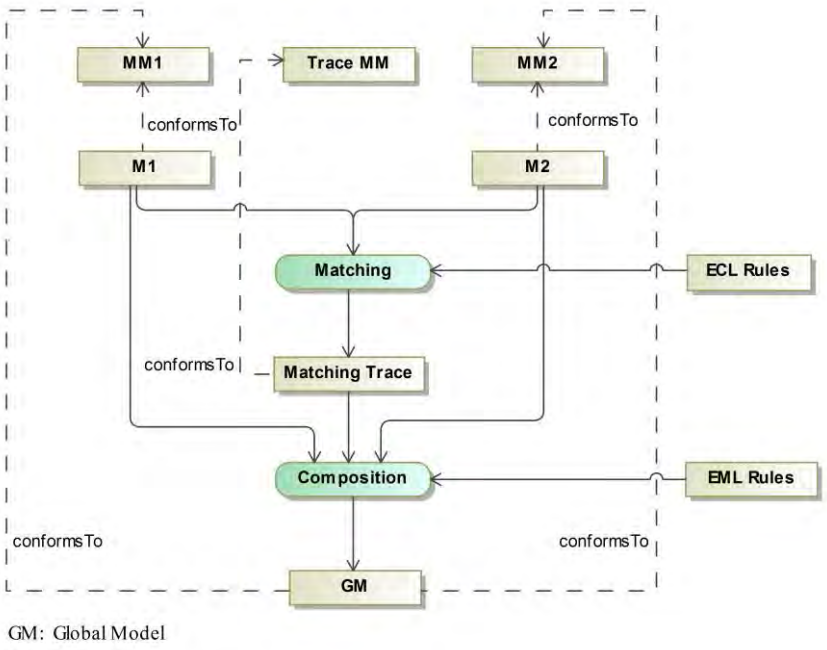


Figure III-18 : Processus de composition d'EML

III.3.2.3.1. ***Phase de mise en correspondance***

En EML, les comparaisons entre modèles sont réalisées grâce à des règles de comparaison entre méta-classes, écrites dans le langage ECL (cf. section III.2.3.6). Le résultat de l'exécution des règles ECL est sauvegardé dans une trace de correspondance. Les données de cette trace sont classées en quatre catégories qui serviront dans la suite du processus de composition (Kolovos et al., 2006b) :

- catégorie 1 : Les éléments qui se correspondent et qui sont conformes,
- catégorie 2 : Les éléments qui se correspondent mais ne sont pas conformes,
- catégorie 3 : Les éléments qui ne se correspondent pas mais sont inclus dans un domaine de comparaison (regroupant l'ensemble des modèles qu'on désire comparer),
- catégorie 4 : Les éléments qui ne se correspondent pas et sont exclus du domaine de comparaison (aucune règle de comparaison à appliquer).

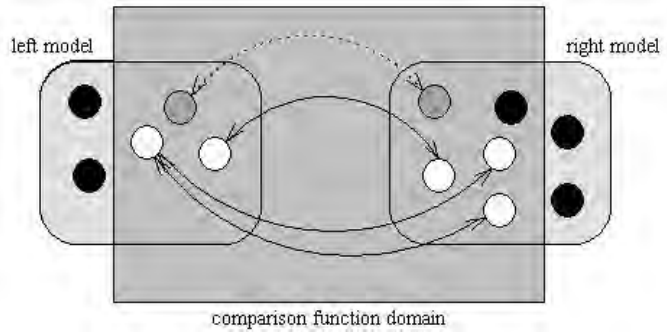


Figure III-19 : Schéma des catégories utilisées en EML (Kolovos et al., 2006b)

Schématiquement, ces catégories peuvent être représentées selon la Figure III-19. Les nœuds reliés par des arcs complets représentent les éléments de la catégorie 1 tandis que ceux reliés par des arcs en pointillé représentent les éléments de la catégorie 2. Le rectangle au milieu représente le domaine de comparaison. Par conséquent, les nœuds non reliés (de couleur noire) à l'intérieur de ce rectangle représentent des éléments de la catégorie 3 tandis que ceux en dehors du rectangle, représentent les éléments de la catégorie 4.

Le Tableau III-2 liste les actions qui sont effectuées en fonction de chaque catégorie à l'exception de la catégorie 4 où le processus se limite à signaler un avertissement. Les éléments de cette catégorie peuvent provenir d'une opération de correspondance incomplète (des règles mal écrites par exemple) ou sont des éléments ignorés intentionnellement (Kolovos et al., 2006b).

Correspondance	Conformité	Interprétation
0	0	Catégorie 3 : Une opération de transformation est réalisée avec ETL
0	1	Aucune catégorie
1	0	Catégorie 2 : Le processus de composition est interrompu à cause d'un conflit détecté
1	1	Catégorie 1 : Une opération de fusion est réalisée avec EML

Tableau III-2 : Tableau récapitulatif des catégories et leurs interprétations par le moteur d'EML

EML fournit un mécanisme de réutilisation sous forme d'héritage donnant aux règles la capacité d'être étendues afin d'être réutilisées par d'autres sans avoir à être recopiées.

III.3.2.3.2. ***Phase de composition***

Cette phase permet de réaliser deux types d'activité selon le résultat de la trace de correspondance de la comparaison précédente : Transformation et fusion. Les éléments des catégories 1 et 2 sont fusionnés alors que ceux des catégories 3 et 4 sont transformés. Les règles de fusion sont appliquées quand une correspondance est identifiée entre des éléments de modèle. Une règle de fusion est définie par un nom, deux paramètres en entrée représentant les paires d'éléments à fusionner et au minimum un élément à produire dans le modèle cible.

A contrario, les règles de transformation sont appliquées quand il n'existe pas de correspondance dans le modèle opposé. La structure de celles-ci est similaire à celles des règles de fusion à l'exception du nombre de paramètres en entrée, réduit à un dans ce cas.

L'existence parallèle de ces deux types de règles est justifiée par le besoin d'avoir un modèle global qui doit contenir l'ensemble des éléments des deux modèles d'entrée. Les éléments qui correspondent sont fusionnés alors que les autres sont transformés.

EML introduit aussi la notion de stratégie. Les stratégies sont des règles attachées à EML comme étant des algorithmes dits *pluggables*. Ce sont des unités de code interfaçables avec le programme de spécification de la composition offrant ainsi l'avantage de réduire le nombre de règles. Un exemple d'utilisation de ces stratégies est la composition de modèles homogènes dont la logique peut être déduite de la structure du métamodèle (à travers plusieurs scénarii de composition). Par conséquent, il n'est pas obligatoire de définir manuellement la totalité des règles complétées par ces stratégies.

III.3.2.4. Multi Document Integration

L'approche *Multi Document Integration* (MDI) présentée dans (Königs et Schürr, 2006) propose une approche de composition de modèle hétérogènes en généralisant le formalisme TGG avec le concept de grammaire multi-graphes.Le processus de composition est illustré par le schéma suivant (Figure III-20).

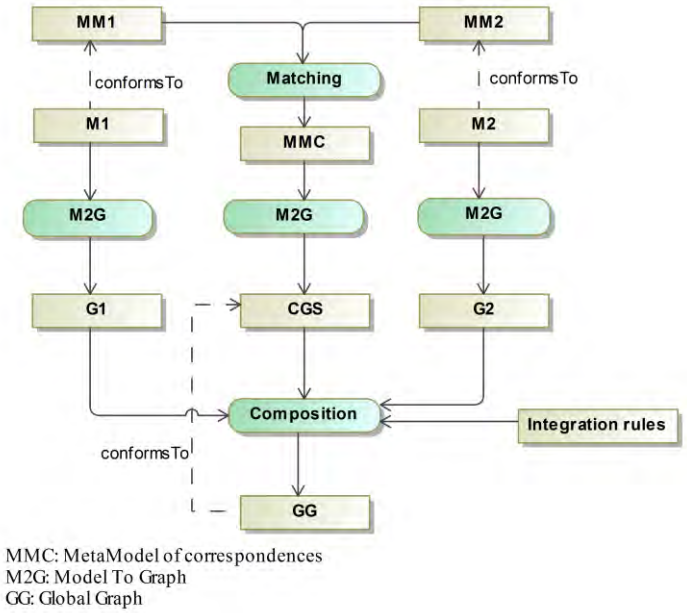


Figure III-20 : Processus de composition de MDI

III.3.2.4.1. ***Phase de mise en correspondance***

L'objectif de la phase de mise en correspondance dans l'approche MDI est la création d'un métamodèle de correspondance. Ce métamodèle est créé graphiquement. Il relie les méta-éléments en les important des différents métamodèles. Les noms attribués aux types de relation sont arbitraires. Ces derniers sont marqués par le stéréotype «*Integration*».

III.3.2.4.2. ***Phase de composition***

Pour permettre l'opération de composition, les modèles sont transformés en schémas de graphes. Par la suite des règles dites d'intégration sont définies pour chaque type de relation existant dans le schéma de graphe de correspondances. La spécification de ces règles est effectuée en utilisant une grammaire multi-graphe qui, contrairement à TGG (Schürr, 1995) (*Triple Graph Grammars*) qui est un formalisme déclaratif d'intégration de graphes, permet la composition d'un nombre arbitraire de graphes. Par la suite, à partir de ces règles, sont dérivées un ensemble de règles opérationnelles qui assurent la cohérence entre les modèles. Ces règles permettront de gérer la propagation des changements effectués sur les éléments de modèle.

III.3.2.5. ATLAS Model Weaver

Atlas Model Weaver (AMW) (Del Fabro et al., 2005) est un Framework développé par l'INRIA permettant la composition de modèles par tissage.

Le schéma de la Figure III-21 présente le processus de composition d'AMW.

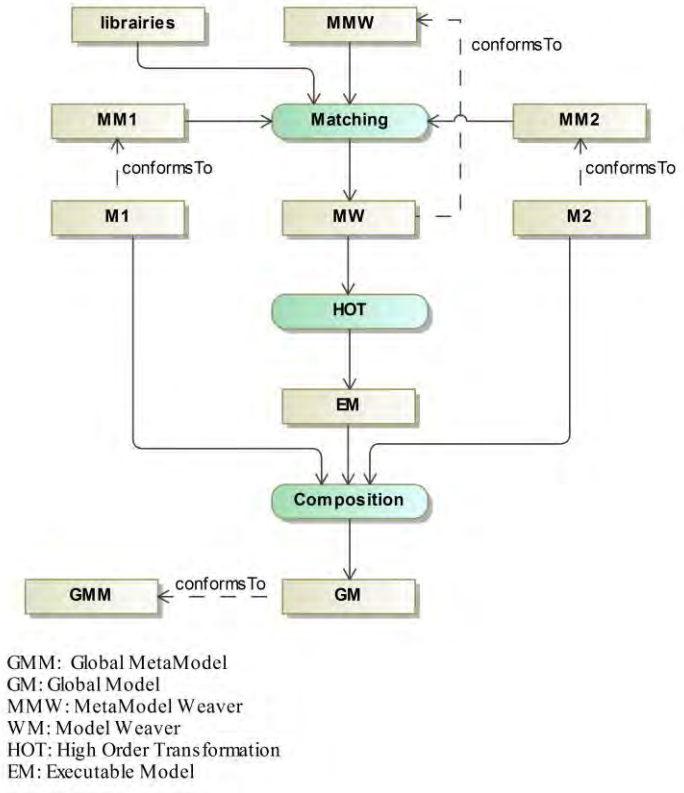


Figure III-21 : Processus de composition d'AMW

III.3.2.5.1. ***Phase de mise en correspondance***

Le mécanisme de mise en correspondance s'effectue en deux étapes. La première étape consiste à définir le métamodèle de correspondance, car le métamodèle de correspondance proposé par défaut par AMW ne contient aucun type de relation. La deuxième étape a pour objectif la création d'un modèle de tissage contenant les correspondances entre les différents éléments des modèles d'entrée (voir section III.2.3.7).

III.3.2.5.2. ***Phase de composition***

Cette étape commence par la création d'un modèle exécutable. Comme l'illustre la Figure III-22, le modèle de tissage obtenu à l'issu de la mise en correspondance est traduit de façon automatique en un modèle de transformation écrit dans un langage d'exécution (ATL a été adopté pour cette tâche), par le biais d'une tierce transformation appelée transformation HOT (*Higher Order Transformation*). Cette transformation est fondée sur des patrons de transformation génériques où les correspondances sont traduites en règles ATL (Del Fabro et Valduriez, 2009).

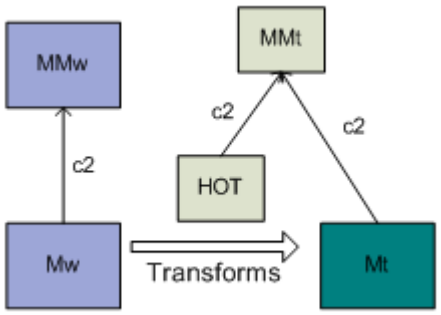


Figure III-22 : Production du modèle exécutable (Fabro, 2008)

Une transformation d'ordre supérieur HOT est une transformation dont l'un des modèles d'entrée ou de sortie (ou les 2) est un modèle de transformation (Del Fabro et Valduriez, 2007). Selon les auteurs (Del Fabro et Valduriez, 2009), le patron de transformation transforme les différents types de relation en des liens exécutables exprimés dans un langage de transformation permettant ainsi la création du modèle exécutable (EM : Executable Model).

Le modèle global final (GM) est obtenu en appliquant les règles de transformation, contenues dans le modèle de transformations exécutables de l'étape précédente sur les modèles d'entrée.

III.3.2.6. VirtualEMF

La majorité des approches de composition étudiées dans les sections précédentes génèrent le modèle composé en y copiant (physiquement) les éléments de modèles à partir des modèles source. Une fois le modèle composé créé, les modèles qui ont servi à sa création n'ont plus d'importance, étant donné que l'utilisateur va agir directement sur le modèle composé et que la manipulation des éléments du modèle est faite directement dans ce dernier.

Ces approches présentent deux inconvénients majeurs concernant :

- le temps de création du modèle composé et la consommation de mémoire dans le cas de composition de modèles de grande taille,
- l'absence de synchronisation, ce qui conduit à exécuter le processus de composition à chaque fois qu'il y a une modification dans les modèles source.

L'approche VirtualEMF (Clasen et al., 2011b) permet de contourner ces limitations en proposant la notion de modèle virtuel.

Contrairement à un modèle concret, un modèle virtuel ne contient pas de donnée physique : il est défini comme un modèle qui redirige les demandes d'accès et de manipulation directement vers les modèles source à partir desquels il a été généré (Clasen et al., 2011a).

Cette méthode de virtualisation présente les avantages suivants :

- une meilleure synchronisation étant donné que les modèles d'entrée et le modèle composé partagent les mêmes instances d'éléments de modèles,
- une création rapide du modèle composé et une utilisation réduite de la mémoire, puisqu'il ne contient pas de donnée concrète.

La Figure III-23 présente le processus de composition de VirtualEMF.

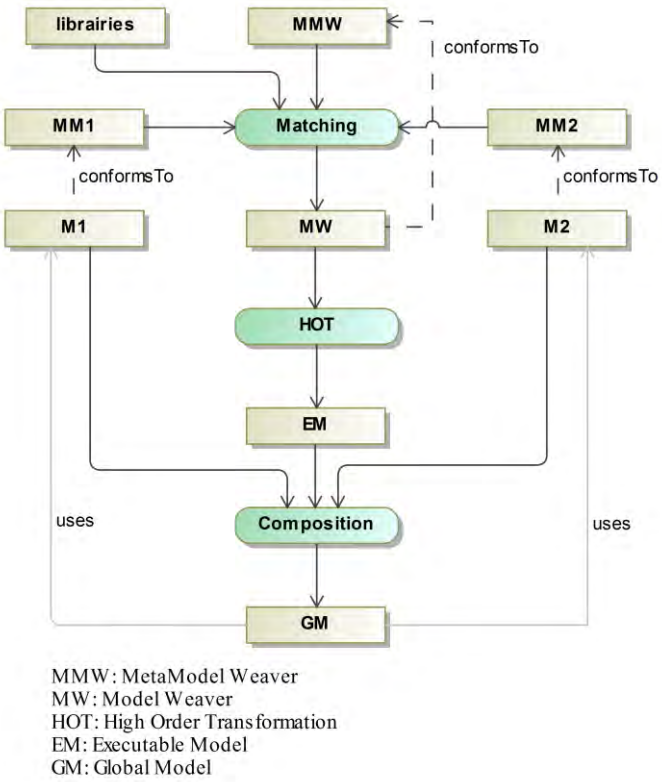


Figure III-23 : Processus de composition de VirtualEMF

Ce processus de composition est similaire à celui proposé dans (Del Fabro et Valduriez, 2009) et décrit par la Figure III-21. La différence réside dans l'ajout de la notion de modèle virtuel. Les différents accès au modèle virtuel sont traduits par des opérations sur les modèles d'entrée (principe de délégation). Deux APIs ont été réalisées pour permettre d'accéder aux modèles de façon transparente : une API de virtualisation (*Virtulization API*) et une API de gestion de liens (*Linking API*). L'exploitation de ces APIs est illustrée sur la Figure III-24.

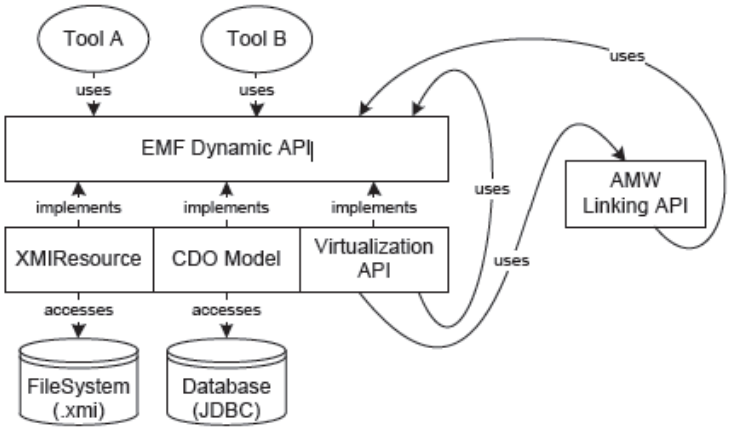


Figure III-24 : Les APIs de VirtualEMF (Clasen et al., 2011a)

III.3.2.6.1. ***Phase de mise en correspondance***

Dans cette phase, les correspondances entre les modèles d'entrée sont établies (en utilisant l'interface graphique d'AMW) afin de produire le modèle de tissage (*Model Weaver*). Les types de relations sont définis en étendant le métamodèle de tissage. La section III.2.3.7 (AMW) a détaillé les étapes de cette phase.

III.3.2.6.2. ***Phase de composition***

La deuxième phase consiste à créer le modèle virtuel en invoquant une fonction (*load*) de chargement des ressources. Cette fonction reçoit en entrée un fichier (*virtualmodel*) de persistance qui stocke les emplacements physiques des ressources incluses dans ce processus : le modèle de tissage, les modèles d'entrée et leurs métamodèles respectifs.

Le chargement est suivi par l'établissement des liens implicites entre les modèles d'entrée et le modèle composé par VirtualEMF. Ces liens permettent de manipuler de façon transparente les éléments du modèle composé.

L'accès à un élément (virtuel) du modèle composé est pris en charge par l'API de virtualisation. Si cette API détecte que cet élément est référencé dans le modèle de tissage alors elle délègue la gestion de cet élément à l'API de liens, dont le rôle est de gérer les différents types de liens puis de retourner le résultat à l'API de virtualisation. Si ce n'est pas le cas, l'API de virtualisation accède directement à l'élément approprié dans le modèle source.

III.3.2.7. Approche de Wouters et al.

L'approche proposée par Wouters et al. (Perin et Wouters, 2014) permet de réaliser la fusion syntaxique et sémantique de différents DSMLs dans l'optique de construire un unique langage de modélisation.

Le processus de composition est le suivant (Figure III-25) :

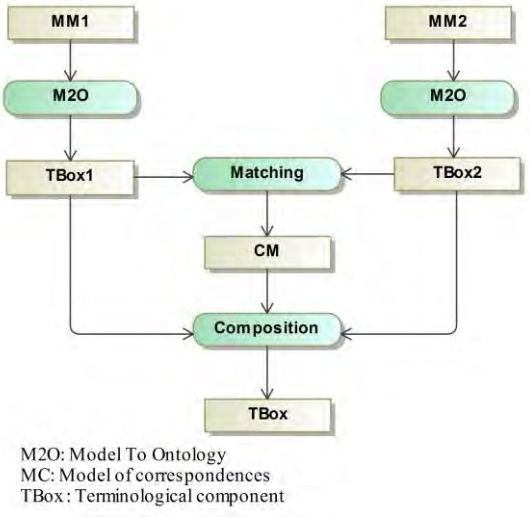


Figure III-25 : Processus de composition de l'approche Wouters et al.

L'opération de mise en correspondance commence par une opération de transformation appelée *lifting* selon (Kappel et al., 2007). Elle consiste à basculer de l'espace technologique des modèles à celui des ontologies (Bézivin, 2006). Dans le cas de l'approche proposée, ceci consiste à transformer les syntaxes abstraites des DSMLs représentées en xOWL. xOWL est une extension de OWL2 munie d'un langage d'action. Son architecture est décrite, selon différents niveaux de modélisation, par la Figure III-26 ci-dessous.

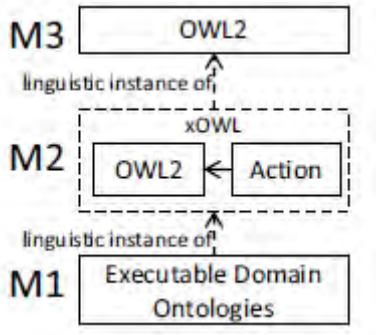


Figure III-26 : Architecure de xOWL (Wouters et Gervais, 2011)

III.3.2.7.1. ***Phase de mise en correspondance***

Cette phase réutilise les techniques de mise en correspondance utilisées dans le domaine des ontologies. Parmi les techniques intégrées dans l'approche de Wouters et al, on peut citer IF-MAP (Kalfoglou et Schorlemmer, 2002) et PROMPT (Noy et Musen, 2000).

III.3.2.7.2. ***Phase de composition***

La composition est facile à mettre en place puisque tous les DSMLs sont traduits dans le même formalisme. La composition revient donc à composer des ontologies (traduites des différents DSMLs) en une seule ontologie globale. Elle se base sur un algorithme, modifiable, défini en langage d'action xOWL. Les modèles sont par la suite créés à partir de l'ontologie globale. Cette approche est dite «notationnelle» (Dupuy-Chessa et al., 2014). Elle permet, en utilisant xOWL, d'avoir pour un élément donné plusieurs notations (syntaxes visuelles) adaptées aux besoins des experts de chaque domaine métier.

III.3.2.8. Approche de fédération de modèles

L'approche de fédération de modèle (Guychard et al., 2013), mise en œuvre sous Openflexo (Agilebirds, 2011), autorise la création de modèles, qui seront composés, à partir d'un ensemble de modèles. La Figure III-27 illustre l'architecture de l'approche où deux modèles virtuels sont créés à partir de trois modèles en entrée. Il est possible de définir une syntaxe graphique pour chaque modèle virtuel grâce à la couche de modélisation.

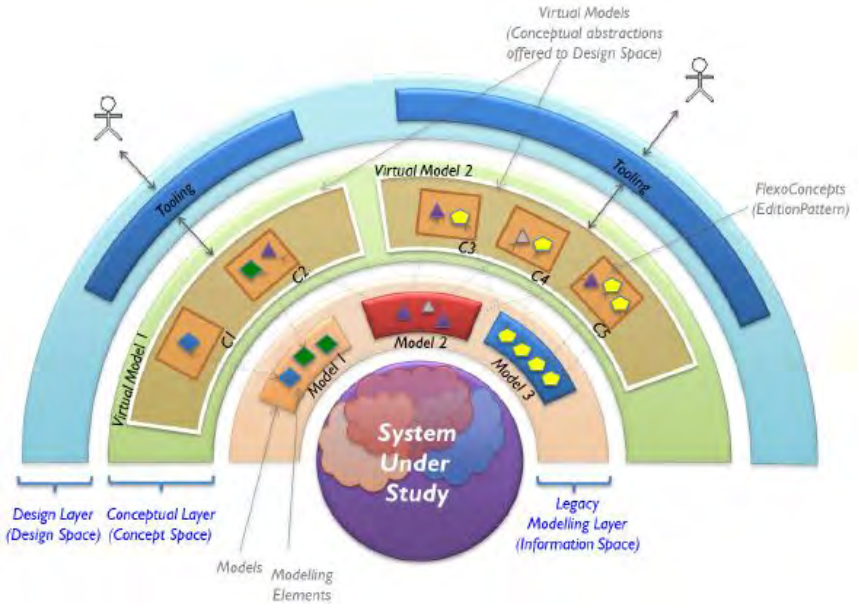


Figure III-27 : Illustration de l'approche de fédération de modèles (Guychard et al., 2013)

La Figure III-28 présente le processus de composition associé à cette approche.

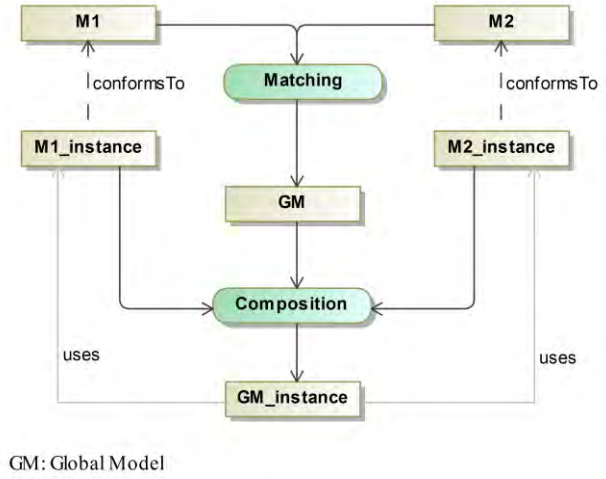


Figure III-28 : Processus de composition de l'approche de fédération de modèles

L'objectif de la mise en correspondance est de définir les concepts fédérés (*federated concepts*) qui seront utilisés lors de l'opération de composition pour relier les instances des éléments de modèles via un mécanisme de virtualisation semblable à VirtualEMF.

III.3.2.8.1. ***Phase de mise en correspondance***

Durant cette phase, l'expert de domaine définit les concepts fédérés qui sont intégrés par l'intermédiaire de l'expert d'outillage dans un modèle virtuel. Ces derniers définissent les correspondances entre les éléments de modèles et sont appelés «patrons d'édition» (*EditionPattern*) sous Openflexo.

La Figure III-29 montre un patron d'édition appelé *City* défini entre deux éléments de type classe. Une contrainte, définie sous forme d'expression conditionnelle, exprime que les deux éléments (ville) sont en correspondance lorsqu'ils ont le même nom (en ignorant la casse) :

cityInModel1.name.toUpperCase = cityInModel2.name.toUpperCase

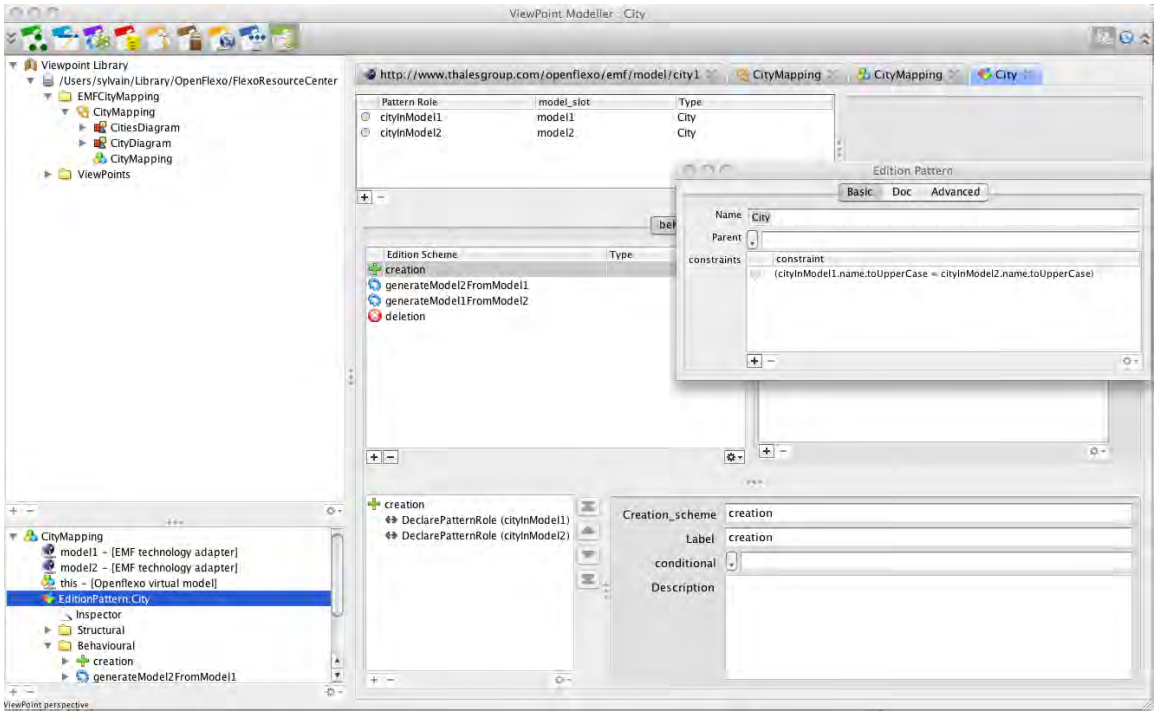


Figure III-29 : Exemple de concept fédéré (*City*) (Guychard et al., 2013)

Une fois les concepts fédérés définis, il est possible de décrire dans le modèle virtuel des procédures de synchronisation. Pour cela il faut définir le modèle maitre et le modèle esclave et la condition à mettre en place. Tout cela doit être attaché au patron d'édition.

Par exemple, la condition représentée dans la Figure III-30 (fenêtre Edition Pattern), consiste à créer, lors de l'ajout d'une ville dans le premier modèle (*model1*), son correspondant au niveau du second modèle (*model2*) s'il n'existe pas.

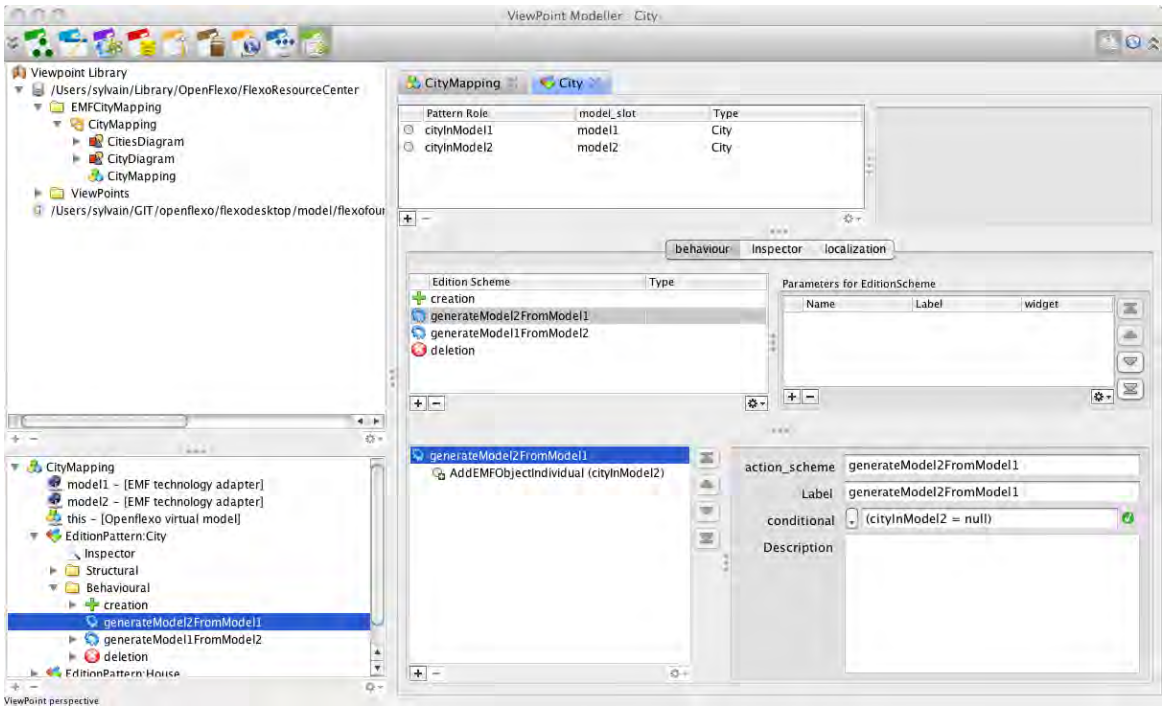


Figure III-30 : Exemple de procédure de synchronisation (Guerin, 2013)

III.3.2.8.2. *Phase de composition*

La phase de composition consiste à instancier le modèle virtuel. Cette instanciation permet la création d'un autre modèle virtuel (VM_0) contenant les correspondances entre les éléments des modèles de niveau M0 impliqués.

La Figure III-31 explicite un exemple de modèle virtuel produit. En sélectionnant le patron d'édition *City* (créé dans la phase précédente), Openflexo affiche toutes les villes. Les noms de ville suivis par [sync] représentent les éléments présents dans les deux modèles et qui valident le patron d'édition *City*. Ces éléments sont donc reliés par une correspondance, comme par exemple Brest et Lampaul-Plouarzel. Les autres éléments de modèles sont présents dans l'un des deux modèles.

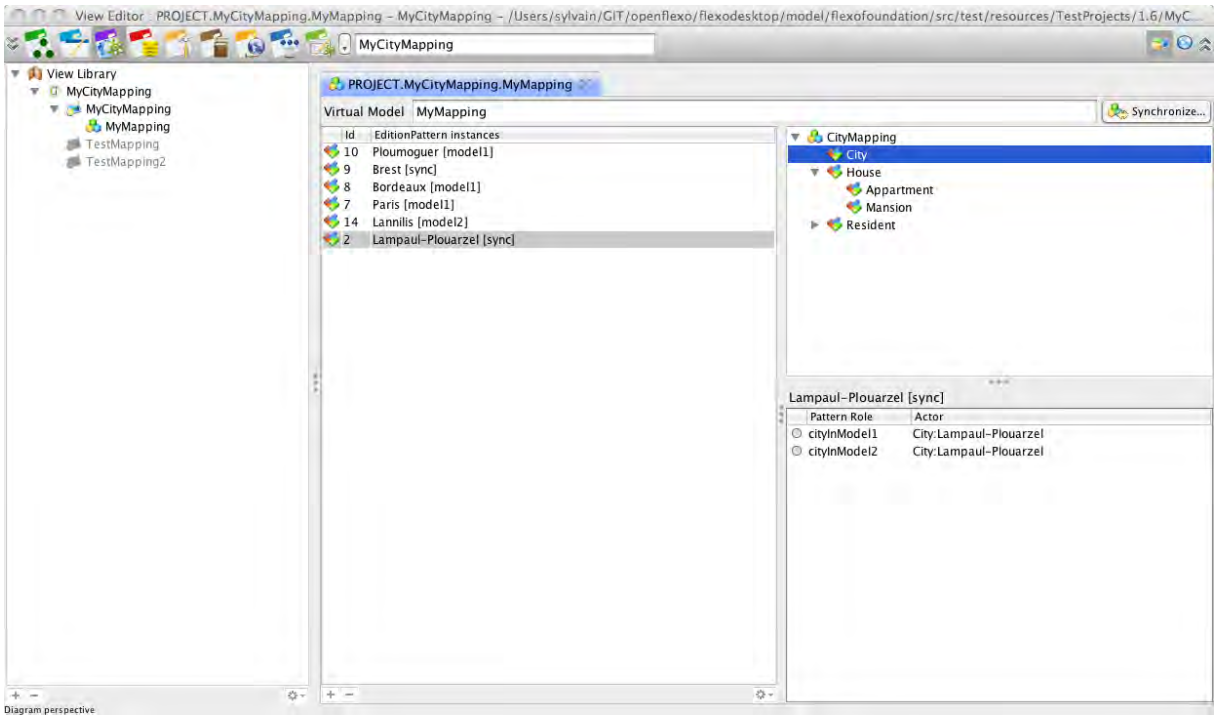


Figure III-31 : Exemple de modèle virtuel produit (MV_0) (Guerin, 2013)

III.3.3. Discussion

Dans cette section, nous analysons les approches présentées ci-dessus en résumant leurs principales caractéristiques.

III.3.3.1. Critères de comparaison

L'objectif de la présente analyse n'est pas d'évaluer les approches en tant que telles, mais de savoir si les critères définis ci-dessous sont couverts par elles.

Les critères retenus sont les suivants :

- Hétérogénéité : Ce critère, déjà exploité précédemment (section III.2.4), indique que le processus de mise en correspondance peut prendre en entrée des modèles hétérogènes. Nous nous intéressons à des modèles issus de différents métiers, qui représentent la réalité de la multi-modélisation des systèmes complexes,
- Nombre d'artefacts en entrée : Ce critère permet de caractériser la limitation éventuelle du nombre d'artefact en entrée,
- Type de composition : D'après notre état de l'art, la composition de modèles est un terme générique qui englobe les notions suivantes : fusion, tissage et virtualisation,
- Préservation des modèles : Il s'agit ici de vérifier que les modèles d'entrée n'ont pas été altérés à la suite de l'exécution de l'opération de composition. Nous nous intéressons aux approches non intrusives,

- Gestion des conflits : Ceci permet d'évaluer la façon dont une approche de composition procède pour garantir la cohérence de l'ensemble des modèles manipulés lors du processus de composition. La gestion de conflits peut être invoquée en pré-composition ou en post-composition. Dans le premier cas, la cohérence des modèles doit être vérifiée avant le processus de composition, tandis que dans le deuxième cas la vérification est faite sur le modèle composé,
- Support visuel : Il s'agit de savoir si l'approche propose une assistance visuelle (graphique ou textuelle),
- Synchronisation : Ce critère qualifie la capacité de l'approche à propager dans le modèle composé les changements effectués dans les modèles d'entrée. Le choix de ce critère est justifié par le besoin d'éviter la réexécution du processus de composition à chaque altération d'un élément de modèle en entrée, et la nécessité d'impacter les modifications directement sur le modèle composé.

III.3.3.2. Bilan des approches étudiées

Critères / Approches	Hétérogénéité	Nbr. d'artefacts en entrée	Type de composition	Préservation des modèles	Gestion des conflits	Support visuel	Synchronisation
AMW	Oui	2..n	Tissage	Oui	Semi-automatique (modèle de tissage et heuristiques)	Graphique	Non
EML	Oui	2..n	Fusion	Oui	Manuelle (à base de règles)	Textuel	Non
Fédération de modèles	Oui	2..n	Virtualisation	Oui	Semi-automatique (à base de patron d'édition)	Graphique	Oui
Kompose	Non	2	Fusion	Non	Semi-automatique (à base de nom et directives)	Textuel	Non
MDI	Oui	2	Fusion	Oui	Manuelle (à base de règles)	Textuel	Oui
UML2PM	Non	2	Tissage	Non	Automatique (à base de nom et contraintes OCL)	Non	Non
VirtualEMF	Oui	2..n	Virtualisation	Oui	Semi-automatique (modèle de tissage et heuristiques)	Graphique	Oui
Wouters et al.	Oui	2..n	Fusion	Oui	Semi-automatique (à base de règles)	Non	Non

Tableau III-3 : Tableau récapitulatif des approches de composition

Nous résumons notre étude comparative dans le Tableau III-3 ci-dessus. Après avoir analysé les différentes approches dans le domaine de la composition, on peut remarquer qu'il y a une corrélation entre la puissance de l'approche et sa complexité : les approches dont le résultat est satisfaisant sont difficiles à manipuler et à exécuter, nécessitant dans la majorité des cas une intervention humaine, qualifiée, tout au long du processus ; les approches plus simples à mettre en œuvre ne produisent pas de résultat aussi précis.

La mise en œuvre de l'approche AMW est une solution fastidieuse qui s'adresse à des utilisateurs confirmés ayant des connaissances permettant d'étendre le métamodèle pour ajouter de nouveaux concepts. Selon (Guychard et al., 2013), cette approche aborde la métamodélisation de façon restrictive et n'offre pas la capacité de définir des correspondances entre les éléments provenant de différents domaines métiers. L'approche EML est similaire à AMW dans le sens où il y a une pré-phase d'établissement des correspondances (règles de correspondances /modèle de tissage) et l'utilisation de langages de programmation supplémentaires.

L'approche Kompose est adaptée uniquement aux modèles structuraux : actuellement cette approche ne gère que les modèles de classes (Acher et al., 2010). De plus le concepteur doit paramétrer le processus de mise en correspondance en définissant les signatures au niveau métamodèle dans le but de définir des opérateurs de correspondance spécifiques. Le concepteur doit définir les directives nécessaires pour pallier les différentes incohérences identifiées.

Le mécanisme de fusion de paquetages UML2PM définit la façon dont le contenu d'un paquetage est étendu avec le contenu d'un autre. De ce fait, le modèle composé regroupe l'ensemble des propriétés des deux modèles : aucun filtre n'est appliqué sur le contenu du modèle composé. Ce mécanisme ne possède pas de base théorique solide, étant donné qu'il est fondé uniquement sur une correspondance à base de noms et ne traite pas les types de relation avec une sémantique plus élaborée. En ce qui concerne la gestion des conflits, les propositions de résolution des incohérences restent limitées et ne correspondent pas aux besoins de l'utilisateur. L'OMG a limité la description de la sémantique de la fusion de paquetages uniquement aux éléments structuraux.

La proposition de Wouters et al. consiste à composer plusieurs DSMLs afin que les experts des différents domaines puissent produire des modèles adaptés à leurs connaissances et à leurs langages respectifs. Cette approche implique que les modèles soient créés après le processus de composition et ne prend donc pas en compte des modèles pré-existants.

Un inconvénient que les approches citées ci-dessus ont en commun est l'élaboration d'un modèle final pour chaque paire de modèles. Ceci rend la gestion des modèles produits difficilement gérable lorsque plusieurs modèles en entrée sont présents.

Les approches de fédération de modèles et de VirtualEMF ciblent une thématique, à laquelle nous nous intéressons, qui consiste en la création d'un modèle global par virtualisation. Etant donné que l'approche VirtualEMF est une variante de l'approche AMW à laquelle les auteurs ont ajouté le mécanisme de virtualisation, elle possède les mêmes limitations qu'AMW. L'approche de fédération de modèles rejoint notre proposition sur plusieurs axes parmi lesquels : la construction d'un modèle virtuel et la synchronisation des changements. Concernant la première proposition, pour chaque concept fédéré (type de relation dans notre approche), une sémantique est définie afin de permettre la création des correspondances. Cependant les sémantiques sont de type structurel uniquement, se basant sur les opérateurs de comparasion. Parallèlement, lorsque les modèles (de niveau M1) changent, les sémantiques des concepts fédérés

doivent être redéfinies étant donné qu'elles sont fortement reliées aux éléments de modèle et non applicables à n'importe quel élément de modèle (de niveau M1). Nous traiterons les inconvénients par rapport à la seconde proposition dans la section III.3.4.6.2.

Il y a aussi une grande variabilité dans la maturité de ces approches. Certaines d'entre elles sont outillées et peuvent donc être réellement exploitées (exemple : AMW, fédération de modèles) alors que d'autres ne proposent que des prototypes ou des solutions théoriques.

III.3.4. Evolution de (méta)modèles et propagation des changements

III.3.4.1. Introduction

L'évolution de (méta)modèles peut survenir à n'importe quel moment du cycle de vie d'un projet. Elle peut avoir différents niveaux d'impact en fonction du type du ou des éléments concernés. Certains auteurs tels que (Favre, 2003) affirment que les langages de modélisation sont stables et ne nécessitent plus (ou rarement) d'évolution. Ce n'est malheureusement pas le cas dans notre contexte. En effet, les évolutions concernent à la fois les GPLs et les DSLs. Un GPL, tel qu'UML par exemple, a subi de nombreuses évolutions en passant d'une version à une autre (la dernière version diffusée par l'OMG, UML 2.4.1 date d'août 2011 (OMG, 2011c)). En ce qui concerne les DSLs, ils sont encore plus enclins à changer puisqu'ils doivent être adaptés chaque fois que leur domaine métier change en raison du progrès technologique ou de l'évolution des besoins.

Pendant la phase de modélisation ou de maintenance par exemple, les concepteurs ont tendance à faire évoluer les différents (méta)modèles sur lesquels ils travaillent afin d'ajouter de nouvelles fonctionnalités. Ceci risque d'engendrer des problèmes à partir du moment où les (méta)modèles sont liés et que la modification de l'un d'eux peut entrainer l'incohérence du système en entier, d'où la nécessité de répercuter les modifications, ou tout au moins d'identifier les éléments de (méta)modèles qui seront impactés par les changements.

L'évolution peut concerner aussi bien les métamodèles que les modèles. Le terme «Adaptation» est utilisé pour les évolutions au niveau métamodèle alors que le terme «Migration» est plutôt utilisé au niveau modèle.

Outre cette notion d'évolution, la littérature définit le principe de coévolution (*coupled evolution*). La coévolution est un cas particulier de l'évolution de modèle. On peut la considérer comme une combinaison d'adaptation et de migration, car elle se produit – entre autres – lorsque l'adaptation d'un métamodèle provoque la migration de ses instances. Donc la coévolution associe à chaque adaptation des règles de migration spécifiques.

On peut classer les évolutions de modèles selon deux niveaux : le niveau vertical et le niveau horizontal. La coévolution se situe au niveau vertical car son objectif est de maintenir la relation de conformité entre le modèle et son métamodèle. Au niveau horizontal, on trouve les évolutions de même niveau. La migration de modèle se positionne donc au niveau horizontal.

Les (méta)modèles peuvent évoluer de différentes manières. Certains changements peuvent être additifs et indépendants des autres éléments de (méta)modèles, ce qui nécessite peu ou pas d'évolution. Dans d'autres cas les manipulations des (méta)modèles peuvent introduire des incompatibilités et des incohérences pas toujours facilement (et automatiquement) résolvables.

Le défi est encore plus grand quand il s'agit de l'évolution de plusieurs modèles. Prenons l'exemple de GMF (Gronback, 2009). C'est un outil de modélisation graphique qui se base sur un ensemble de langages de modélisation liés entre eux, afin de générer le code Java d'un DSL graphique.

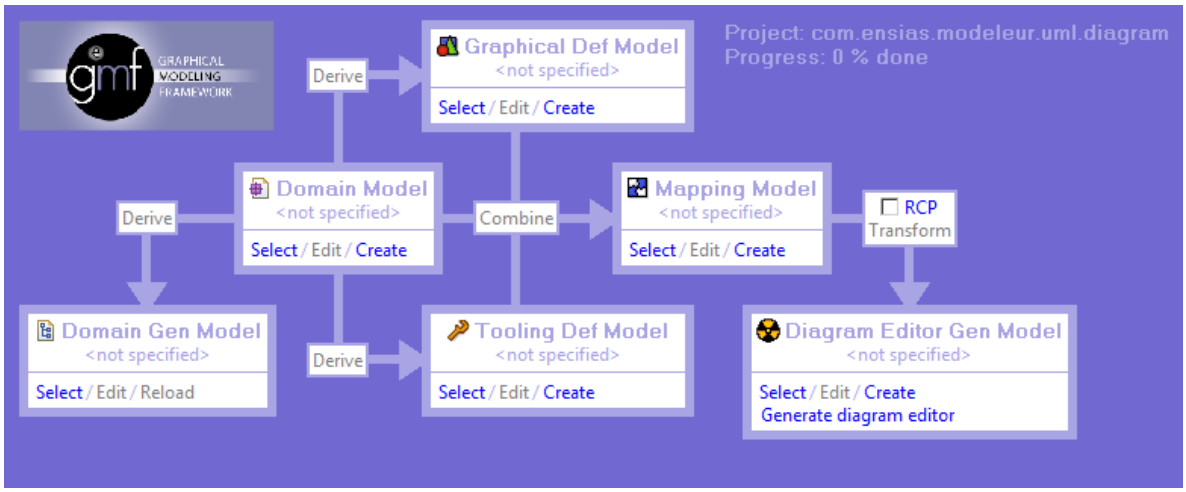


Figure III-32 : Processus de génération d'un éditeur graphique avec GMF (snapshot de notre environnement de travail)

La Figure III-32 illustre les différents modèles intervenant lors de la création d'un projet GMF. La création du modèle de mapping (*Mapping Model*) est faite à partir du modèle de définition graphique (*Graphical Def Model*), du modèle d'outillage (*Tooling Def Model*) et du modèle de domaine (*Domain Model*). La modification du modèle de domaine peut entraîner l'incohérence des autres modèles d'où la nécessité de propager les changements effectués.

Selon (Gruschko et al., 2007), le mécanisme d'évolution de modèles suit trois phases :

- Détection des changements,
- Classification des changements selon l'impact,
- Traitements des impacts des changements et migration des modèles.

III.3.4.2. Détection des changements

Contrairement à la phase de mise en correspondance qui met en évidence les similarités et les points en commun entre les éléments de modèles, le résultat de cette étape de détection des changements est la spécification des discordances (deltas) issues de l'évolution d'un (ou plusieurs) modèle(s). Le delta construit permet par la suite d'identifier les éléments de modèles impactés par le changement et les modifications susceptibles d'être appliquées sur les (méta)modèles afin de maintenir la cohérence du domaine d'application.

Dans cette phase les changements identifiés sont regroupés en trois ensembles disjoints :

- Modification : Un nouvel état des éléments de modèle est défini suite à des mises à jour d'éléments existants,
- Ajout : De nouveaux éléments de modèle sont ajoutés aux modèles initiaux,
- Suppression : Des éléments de modèles n'existent plus en raison d'une opération de suppression.

Nous verrons par la suite que ces trois ensembles sont exploités de façon différente selon l'approche utilisée.

III.3.4.3. Classification des changements

La classification des changements permet de mieux gérer les impacts en affectant à chaque cas un traitement particulier. Gruschko et al. (Gruschko et al., 2007) ont proposé une classification en trois catégories :

- *Non breaking changes* : Cette catégorie correspond aux évolutions dont les changements sont de type «non cassant» : l'impact de ce type de changement est nul et ne met pas en cause la cohérence du système. Exemple : ajout d'une nouvelle (méta-) classe ou d'une nouvelle (méta-)propriété,
- *Breaking and resolvable* : Les évolutions de cette catégorie reflètent des changements de type «cassant» mais qui peuvent être résolus automatiquement sans intervention humaine. Exemple : modification d'un élément de modèle,
- *Breaking and non resolvable* : Cette catégorie fait référence à des changements de type «cassant» qui ne peuvent pas être résolus automatiquement. Une intervention humaine est alors nécessaire pour apporter des informations supplémentaires afin d'assurer l'opération d'adaptation. Dans le cas d'une action sur un métamodèle, une intervention sur ses instances va être nécessaire afin de maintenir le lien de conformité. Exemple : la suppression (ou la modification) d'un élément du métamodèle implique que l'instance concernée soit supprimée. Ceci risque de poser problème à partir du moment où cette instance est reliée à d'autres éléments, etc.

III.3.4.4. Migration de modèles et impacts des changements

Lors de cette phase, des opérations de migration peuvent être appliquées afin de propager les changements (détectés et classifiés précédemment) d'un modèle à un autre. L'opération de migration est une opération de transformation qui concerne les éléments nécessitant un ajustement afin de maintenir la cohérence entre les modèles du domaine d'application lors de leurs évolutions. Comme cela est détaillé dans les sections suivantes, ces éléments sont spécifiés dans un modèle de différence (dans le cas d'une technique de migration à base d'état par exemple). Cependant il n'y a pas de restriction sur les éléments de modèles d'entrée d'une opération de transformation.

Après une évolution de modèle, deux types de mécanismes de migration peuvent être spécifiés pour garantir la transition d'une version de modèle vers une autre : à base d'opération ou à base d'état.

III.3.4.4.1. *Migration à base d'opération*

Les approches de migration dites «à base d'opération» s'appuient sur la sauvegarde des opérations de changement, afin de garder trace des modifications effectuées sur le (méta)modèle. Dans les approches de ce type, il n'est pas nécessaire de retrouver les changements effectués (via un modèle de différence par exemple), puisque les modifications sont enregistrées et stockées directement (Koegel et al., 2009) (il n'existe qu'une seule version d'un (méta)modèle). Dans ce contexte, certaines approches proposent aux utilisateurs des opérations prédéfinies. Pour chaque

opération, une procédure de migration correspondante est fournie afin de réduire l'effort associé à la migration. Cependant ces opérations ne couvrent pas tous les changements. De ce fait, certaines approches comme Edapt (Herrmannsdoerfer, 2011) et COPE (Herrmannsdoerfer et al., 2008) fournissent un moyen de modifier manuellement le (méta)modèle par l'intermédiaire d'un éditeur approprié.

Le principal inconvénient des approches à base d'opération est qu'elles nécessitent la modification du (méta)modèle correspondant avec des outils dédiés implémentant un outil de sauvegarde des changements (*recorder*) ce qui n'est pas toujours réalisable, car les traces des changements liés à la modification du (méta)modèle ne sont pas toujours disponibles.

III.3.4.4.2. ***Migration à base d'état***

Les approches utilisant ce mécanisme, telles que EMFMigrate (Di Ruscio et al., 2011), sont facilement applicables aux (méta)modèles qui ne sont pas considérés comme des artefacts primaires lors d'une opération de modification. Ces approches ne stockent que les états d'un modèle (ajout, modicfication et suppression des élements de modèles). Il faut donc se baser sur une évaluation des différences en comparant les deux états (initial et courant) d'un (méta) modèle avec des outils (ou techniques) de comparaison. Le modèle de différence peut servir par la suite comme entrée dans une opération de transformation (Conradi et Westfechtel, 1998).

Les inconvénients majeurs de ce type d'approche sont :

- La perte de l'ordre temporel des modifications, qui peut s'avérer être un élément important pour comprendre les changements,
- La complexité du calcul des différences entre (méta)modèles est élevée, surtout si les modèles sont de grande taille. Une modification complexe peut entraîner un grand nombre d'états en fonction du type de changement et des relations entre les différents artefacts.

III.3.4.5. Approches de traitement de l'évolution de modèle

Plusieurs approches de la littérature traitent le problème de l'évolution de modèle. Dans cette section nous nous intéressons aux plus représentatives à notre connaissance, à savoir Edapt, EMFMigrate, l'approche de Cichetti et al., et le mécanisme de synchronisation de l'approche de fédération de modèles (section III.3.2.8).

III.3.4.5.1. ***Edapt***

M. Herrmannsdoerfer (Herrmannsdoerfer, 2011) propose une approche de migration de modèles implémentée en Java appelée Edapt. Ce Framework, précédemment appelé COPE (Herrmannsdoerfer et al., 2008), fournit un mécanisme de gestion de coévolution. Tous les changements détectés entre deux versions d'un métamodèle sont enregistrés de façon automatique et des opérations de migration leur sont associées. Edapt fournit aussi des primitives

pour faciliter l'expression des opérations d'adaptation et de migration qui doivent être complétées par la suite par le concepteur.

III.3.4.5.2. ***EMFMigrate***

EMFmigrate est un DSL à syntaxe concrète textuelle dédié en général à la gestion de la coévolution de modèle. Contrairement à la majorité des travaux dans cet axe, EMFMigrate n'est pas limité à un type précis d'artefact de modélisation mais adapté à une large variété d'artefacts (exemple : transformations, syntaxe concrète, etc.). Par exemple, une transformation ATL est un artefact automatiquement adapté avec EMFMigrate à la suite d'une modification sur les éléments de métamodèle source figurant dans les règles de transformation.

Cette approche propose un éditeur permettant (Wagelaar et al., 2012) :

- la spécification des librairies de migration visant à concrétiser et à permettre la réutilisation des règles d'adaptation pour les artefacts récurrents,
- la personnalisation des règles de migration déjà disponibles dans les librairies,
- la gestion des migrations qui ne peuvent pas être entièrement automatisées et qui nécessitent l'intervention de l'utilisateur.

La Figure III-33 décrit la structure d'une spécification de migration sous EMFMigrate.

```
include {library*}
migrate A : MM with Delta {
  rule mr1
     [ guard1 ] {rewritingRule* }
  rule mr2
     [ guard2 ] { rewritingRule* }
  ...
  rule mrn
     [ guardn ] { rewritingRule* }
}
```

Figure III-33 : Structure du programme de migration en EMFMigrate (Di Ruscio et al., 2011)

Chaque spécification de migration est décrite en utilisant un modèle de différence (Delta) qui peut être spécifié manuellement ou généré automatiquement au moyen d'approches existantes de calcul de différence de modèles (par exemple : EMFCompare (Brun et Pierantonio, 2008)). En particulier, un programme de migration est capable de migrer un artefact A, conforme au métamodèle MM, en fonction des différences identifiées dans le modèle Delta, conforme au métamodèle de différence proposée dans (Cicchetti et al., 2007) à travers des règles de migration nommées «mri» (*migration rules*) dans la Figure III-33.

III.3.4.5.3. ***Approche de Cicchetti et al.***

Les auteurs proposent une approche à base de transformations de modèles permettant d'adapter des modèles existants à la suite de modifications effectuées sur un modèle tiers. Cette approche opère par exécution séquentielle de deux transformations (cf. Figure III-34). Elles

consistent à obtenir en premier lieu un métamodèle de différence à partir duquel le modèle de différence est instancié, suivi en second lieu d'un modèle contenant les opérations de migration.

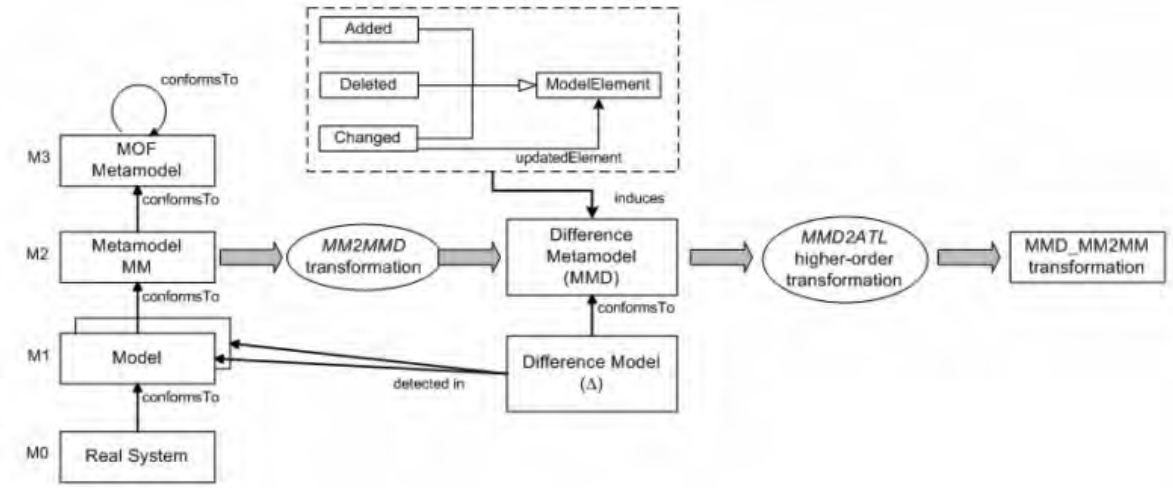


Figure III-34 : Structure générale de l'approche de Cicchetti et al. (Cicchetti et al., 2008)

Définition du modèle de différence

Ce modèle (Δ), conforme à un métamodèle de différence (MMD), permet d'identifier les éléments de modèle auxquels les modifications doivent être appliquées ; les éléments qui ne sont pas affectés par les changements ne sont pas référencés dans le modèle de différence.

Le métamodèle de différence nommé MMD est obtenu en étendant le métamodèle de base (commun car les modèles d'entrée sont homogènes) par application d'une transformation MM2MMD. Pour chaque méta-classe MC du métamodèle d'entrée, les méta-classes *AddedMC*, *DeletedMC*, et *ChangedMC* sont générées. En fait, le métamodèle de différences n'est autre que le métamodèle de base incluant les concepts d'ajout, de suppression et de modification pour chaque méta-classe.

L'introduction de nouvelles méta-classes pour désigner des éléments ajoutés, supprimés et modifiés pour chaque méta-classe n'est pas la seule façon dont la modélisation des changements est réalisée. En fait, dans (Rivera et Vallecillo, 2008), les auteurs affirment que le métamodèle de différence doit être plus général afin d'être indépendant des métamodèles d'entrée. Pour cette raison, ils proposent un métamodèle, non greffé dans les métamodèles d'entrée, composé de 3 méta-classes fondamentales : *AddedElement*, *DeletedElement* et *ModifiedElement*. Ainsi, chaque élément du modèle de différence (de type *DiffElement*) aura pour type l'une des méta-classes de la Figure III-35, selon que l'élément a été ajouté, supprimé ou modifié.

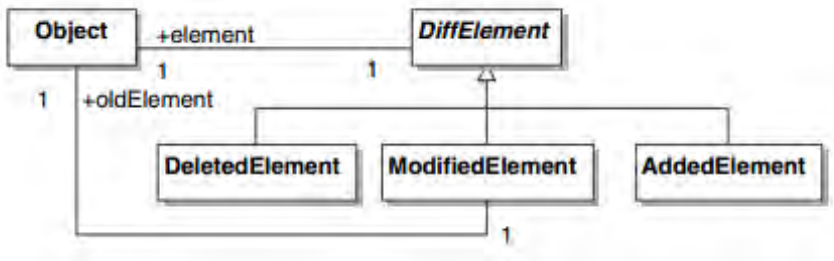


Figure III-35: Métamodèle de différence (Rivera et Vallecillo, 2008)

Opération de migration

La migration est effectuée en exécutant la transformation MMD_MM2MM, qui est générée automatiquement à travers une transformation HOT MMD2ATL (cf. Figure III-34).

Le code ATL de migration prend en entrée le modèle d'entrée ainsi que le modèle de différence et produit un autre modèle en fonction des changements identifiés.

III.3.4.6. Discussion

Cette section propose une évaluation des approches de traitement de l'évolution de modèles détaillées dans la section précédente.

III.3.4.6.1. *Les critères de comparaison*

Pour la comparaison des approches, nous nous basons sur les critères ci-dessous, identifiés à partir de notre état de l'art et de notre besoin en multi-modélisation :

- Hétérogénéité : Comme déjà utilisé plus haut, ce critère indique que le processus de mise en correspondance peut prendre en entrée des modèles hétérogènes,
- Nombre d'artefacts en entrée : Ce critère permet de caractériser la limitation éventuelle du nombre d'artefact en entrée,
- Détection des changements : Il permet d'évaluer comment une approche procède pour détecter les éléments d'artefacts qui ont subi une altération,
- Support de classification : Ce critère indique si l'approche intègre une classification des changements. La classification permet d'associer à chaque type de changement une action particulière. Il est intéressant de prendre en compte ce critère, parce que la classification des changements permet l'automatisation du processus de traitement de l'évolution ou au moins une partie de celui-ci,
- Type d'artefact : Ce critère décrit les artefacts qui peuvent être concernés par l'évolution,
- Langage d'implémentation des règles de migration : Ce critère décrit le langage avec lequel l'approche est mise en place. C'est avec ce langage que le traitement de l'évolution est réalisé.

III.3.4.6.2. ***Bilan des approches étudiées***

Critères / Approches	Hétérogénéité	Nbr. d'artefacts en entrée	Détection des changements	Support de classification	Type d'artefact	Langage d'implémentation
Cichetti et al.	Non	2	Manuelle	Non	Modèle	ATL
Edapt (COPE)	Non	2	Semi-automatique	Non	Modèle	Java (Groovy)
EMFMigrate	Non	2	Manuelle	Non	Modèle/Transformation	DSL spécifique
Fédération de modèles	Oui	2..n	Non-défini	Non	Modèle	openflexo

Tableau III-4 : Tableau comparatif des approches de traitement de l'évolution de modèles

Le Tableau III-4 compare les approches de traitement de l'évolution de modèles présentées dans la section précédente en fonction des critères que nous avons établis. En analysant ces approches on peut déduire que le processus d'évolution n'a pas encore atteint sa maturité. En effet, un certain nombre d'inconvénients demeurent. Premièrement, pour un changement complexe exploitant les transformations HOT (dans l'approche de Cichetti et al.), qui affecte par exemple plusieurs éléments en même temps, de multiples règles ATL risquent d'être produites. Le problème réside dans le choix de l'adaptation appropriée à exécuter, ce qui nécessite la personnalisation des règles et par ailleurs des connaissances techniques. Deuxièmement, les approches étudiées ne définissent aucune pré-phase de classification, ce que nous jugeons obligatoire afin de pouvoir traiter automatiquement certains changements. Troisièmement, la plupart des approches étudiées dans la littérature, exceptées celle de Cichetti et al. et de la fédération de modèles, se concentrent sur la migration de modèles à l'issue de l'adaptation du métamodèle correspondant (principe de coévolution) afin de préserver le lien de conformité entre le modèle et son métamodèle. C'est-à-dire que ces approches privilégient le niveau vertical sans prendre en considération l'évolution horizontale. C'est dans le cadre de ce type d'évolution que s'inscrit la synchronisation de modèles. Quatrièmement, l'approche de Cichetti et al. requiert un métamodèle de différence spécifique et c'est à la charge du concepteur (comme dans EMFMigrate) de détecter les changements et de les représenter au niveau du modèle de différence. Cinquièmement, l'approche de fédération de modèles présentée dans la section III.3.2.8 répond à notre problématique en se basant sur les correspondances établies pour la gestion de la cohérence mais ne propose pas de mécanisme de gestion de la traçabilité des changements. Par ailleurs, l'approche se base essentiellement sur la synchronisation des changements. De ce fait elle ne gère pas les impacts dus à l'ajout ou à la suppression d'éléments de modèles. Pour les changements de type modification, la synchronisation ne peut pas être appliquée sans qu'aient été définis un modèle maître et un modèle esclave. Enfin, les approches présentées ne tiennent pas compte de tous les critères décrits et ne répondent donc pas pleinement à certains aspects importants du traitement de l'évolution.

III.4. Conclusion sur le chapitre

Dans ce chapitre, nous avons examiné la littérature en réalisant une étude bibliographique des travaux relatifs au domaine de la multi-modélisation. De façon générale, les approches de

mise en correspondance étudiées présentent des lacunes sur deux axes : avant et après la création du modèle de correspondance. Concernant le premier axe, nous notons d'abord le manque d'équilibre entre la personnalisation des concepts de mise en correspondance et leur réutilisabilité. Les approches existantes se basent principalement sur l'un des deux critères uniquement (par exemple la réutilisation se fait au détriment de la personnalisation et vice versa). Ensuite, la majorité de ces approches n'exploitent que des correspondances de cardinalité binaire et ne permettent donc pas d'établir des correspondances complexes de type n-aire reliant un élément d'un modèle à un nombre quelconque d'éléments d'autres modèles. De ce fait, les approches nécessitent un modèle de correspondance entre chaque paire de modèles d'entrée (donc pour un nombre n de modèles, [n*(n-1)]/2 modèles de correspondance doivent être produits) ce qui a pour conséquence la création d'un grand nombre de modèles de correspondances séparés, sans lien entre eux, et rend leur gestion difficile et presque impossible à automatiser. D'autre part, les modèles évoluant dans le temps, la modification de l'un d'eux peut entraîner l'incohérence du modèle composé d'où la nécessité de répercuter les modifications, ou tout au moins d'identifier les éléments de modèles qui seront impactés par les changements. Les approches identifiées ne traitent pas complétement cet aspect d'évolution du domaine d'application. Ceci vient du fait que ces dernières n'exploitent pas les correspondances préalablement établies pour offrir un mécanisme qui garantisse la cohérence du modèle global.

Dans la suite de ce manuscrit, nous proposons une approche permettant de répondre aux limitations identifiées ci-dessus.

Le chapitre suivant traite de la mise en correspondance de modèles hétérogènes.

CHAPITRE IV. MISE EN CORRESPONDANCE DE MODÈLES HÉTÉROGÈNES

"In quatum physics, particules even far apart are linked by a ghostly connection.
Connected particles are coordinated as their states are bound
even when they are separated by great distances"

— *Albert Einstein, Boris Podolsk, Nathan Rosen*

IV.1. Introduction

Dans le contexte de notre travail, nous avons étudié plusieurs approches de mise en correspondance. Nous avons constaté dans le chapitre précédent que les approches traitées proposaient des langages de modélisation présentant divers inconvénients. On peut citer entre autres : l'arité des correspondances établie qui est limitée à deux, l'existence généralement d'un seul type de relation (similarité) pour exprimer les correspondances et la pluralité des modèles de mise en correspondance produits qui varient en fonction du nombre de modèles à relier. Pour remédier à cela nous proposons de construire un modèle de correspondance global basé sur un langage ubiquitaire (selon la définition de (Evans, 2004)). Pour cela, nous proposons un mécanisme permettant, dans un premier temps, d'identifier des correspondances entre éléments de métamodèles, puis, dans un deuxième temps, de produire par raffinement des correspondances entre les éléments de modèles. Le mécanisme de raffinement permet ainsi de définir une correspondance une seule fois au niveau métamodèle et de la réutiliser pour les instances d'éléments de métamodèles. En d'autres termes, les correspondances établies au niveau des métamodèles servent de guide pour établir les correspondances au niveau modèles, et permettent de déduire ou d'empêcher la création de certaines autres, en fonction du type de la relation et des éléments à relier (on parle de validation de premier niveau).

Dans notre approche nous distinguons donc les notions de :

- Correspondance entre méta-éléments au niveau métamodèle nommée HLC (*High Level Correspondence*). Elle contient les méta-éléments reliés via un type de relation de haut niveau nommé HLR (*High Level Relationship)*,
- Correspondance entre éléments au niveau modèle nommée LLC (*Low Level Correspondence*). Celle-ci contient les éléments de modèle reliés via un type de relation de bas niveau nommé LLR (*Low Level Relationship)*.

IV.2. Exemple fil conducteur : Système de gestion de conférence

Les processus de production et de relecture de documents sont très utilisés dans divers contextes tels que la gestion de projets, le développement de logiciel, la gestion de conférences scientifiques, etc. Nous nous plaçons plus particulièrement dans ce dernier contexte.

Comme les documents sont maintenant rédigés dans un format électronique, la plupart des communautés scientifiques ont récemment établi des politiques et des mécanismes afin de mettre en œuvre la gestion électronique de conférence, notamment en exploitant l'internet comme infrastructure de communication et de coopération (Papagelis et al., 2005).

Pour illustrer la problématique de notre travail, nous avons choisi un système de gestion de conférence (CMS : *Conference Management System*) comme fil conducteur. Ce système est conçu pour supporter les principales fonctions requises pour la gestion d'une conférence telles que : appel à communication, soumission d'articles, relecture des articles, notification de la décision finale, inscription, etc.

Le choix de cet exemple s'explique d'une part par le fait qu'il est bien connu des chercheurs, d'autre part, par le fait qu'il implique différents acteurs (concepteurs) que l'on peut associer à des modèles hétérogènes.

Nous avons choisi – pour ne pas trop complexifier l'exemple – de représenter le CMS par trois domaines métier. Ces domaines métier sont représentés par des modèles (cf. Figure IV-1) supposés avoir été élaborés séparément par les 3 acteurs suivants :

- Architecte logiciel : responsable de la conception logicielle du CMS ; il produit un modèle de conception logicielle (*Software design*) exprimé à travers un métamodèle spécifique de conception logicielle,
- Architecte de base de données : responsable de la mémorisation des données ; il crée un modèle (*Persistence*) exprimé à travers un métamodèle de persistance,
- Concepteur de procédés : responsable de la façon de mettre en oeuvre un CMS ; il crée un modèle (*Business process*) exprimé à l'aide d'un métamodèle de processus métier.

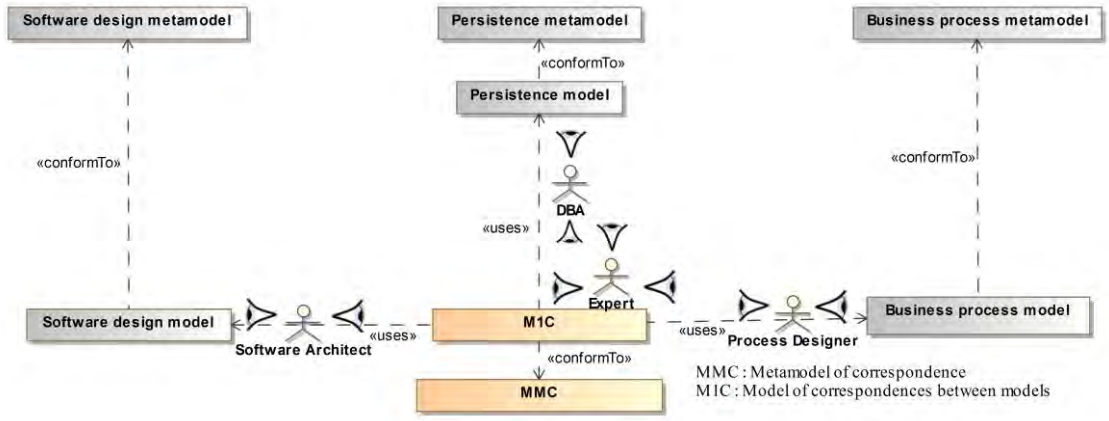


Figure IV-1: Vue globale des modèles du CMS

Dans les sous-sections suivantes, nous présentons les modèles ainsi que les métamodèles associés. La création du M1C sera abordée dans la section IV.7.

IV.2.1. Modèle de conception logicielle

Dans la Figure IV-2, nous proposons un métamodèle, inspiré d'UML, simple mais suffisant pour la description de la conception logicielle. Il définit des entités, avec leurs propriétés et opérations, ainsi que les associations qui existent entre elles.

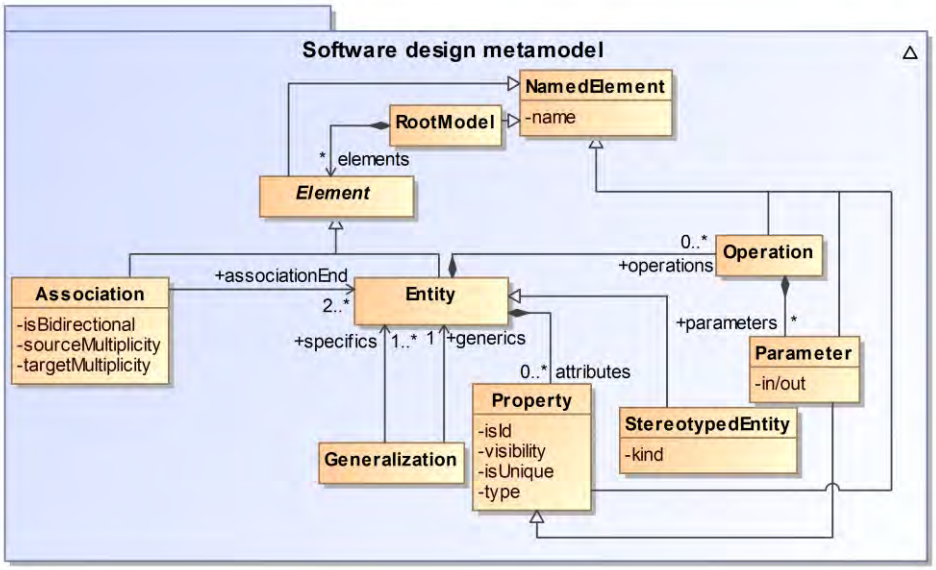


Figure IV-2 : Métamodèle de conception logicielle

La Figure IV-3 illustre un aperçu d'un modèle conforme au métamodèle précédent. Il est constitué de quatre types d'utilisateurs (organisateur, auteur, participant et relecteur) avec leurs informations personnelles. La conférence à gérer est décrite par l'intermédiaire de l'entité *conference*, qui contient différentes propriétés. On peut citer par exemple, le domaine de recherche, la date limite de soumission, la date de notification, le lieu de la conférence, etc. Au sein d'une conférence, plusieurs papiers sont ajoutés via l'opération *addPaper*(). Un papier, représenté via l'entité *paper*, est caractérisé par plusieurs propriétés parmi lesquelles : titre, résumé, mots clé, contenu, etc. et un identificateur attribué une fois que le papier a été soumis. Un papier peut être rédigé par plusieurs auteurs d'où l'assosciation nommée *writes* et il est assigné à plusieurs relecteurs pour évaluation.

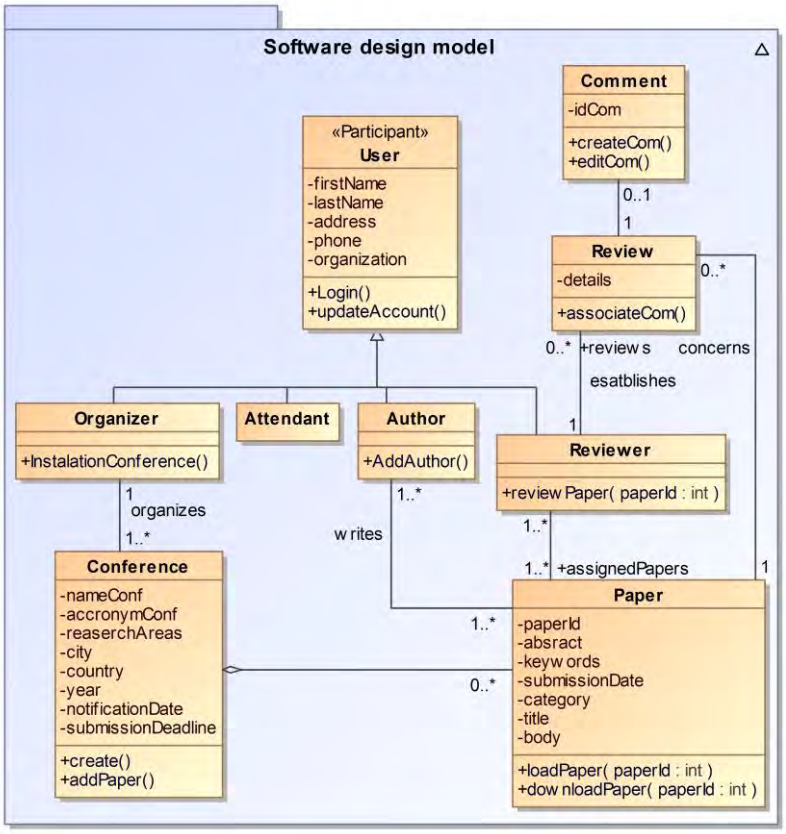


Figure IV-3: Modèle de conception logicielle

IV.2.2. Modèle de persistance

Le métamodèle proposé (cf. Figure IV-4) représente le socle des éléments d'une base de données relationnelle et les relations entre eux. Le schéma de la base de données regroupe des tables et vues. Ces dernières contiennent des colonnes qui peuvent être aussi des clés primaires ou des clés étrangères.

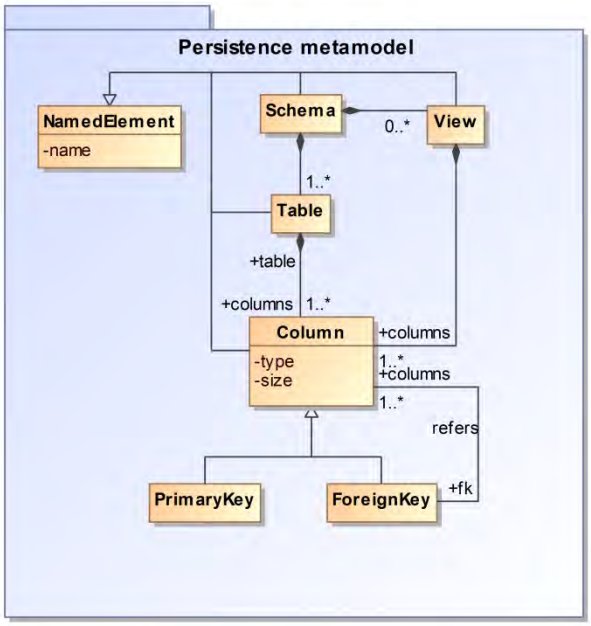


Figure IV-4 : Métamodèle de persistance

La Figure IV-5 présente le modèle créé. Il contient les tables permettant d'enregistrer les informations propres à un acteur (*AuthorTable*) telles que son identificateur, le nom de l'organisation à laquelle il est affilié. La table *ArticleReviewsTable* reflète les données concernant l'évaluation d'un article à savoir une colonne *reviews* pour les remarques et critiques, une colonne *decision* pour renseigner l'acceptation ou bien le refus. Dans la même table, une colonne de type clé primaire est utilisée pour énumérer de façon unique les différentes évaluations ainsi qu'une colonne de type clé étrangère permettant d'identifier le relecteur.

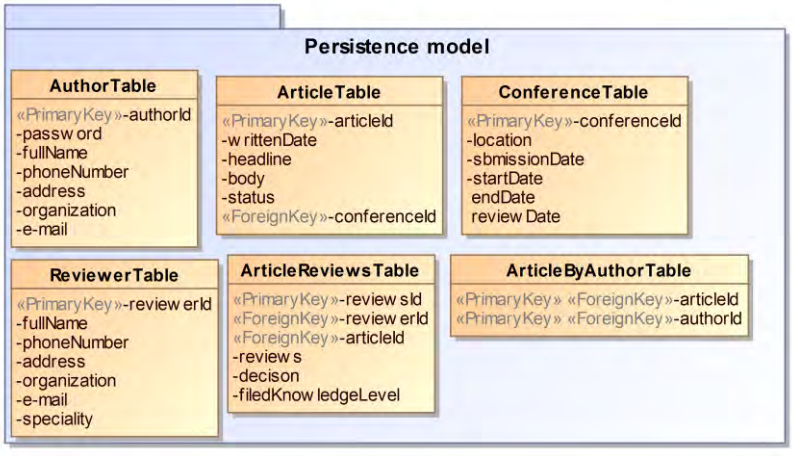


Figure IV-5: Modèle de persistance

IV.2.3. Modèle du processus métier

La gestion d'une conférence peut être vue comme un processus métier que les différents acteurs doivent mettre en œuvre pour garantir son bon déroulement. Le métamodèle choisi (cf. Figure IV-6) est celui de la notation BPM (OMG, 2013). Il comprend les concepts : *lane*, *pool*, *task*, etc.

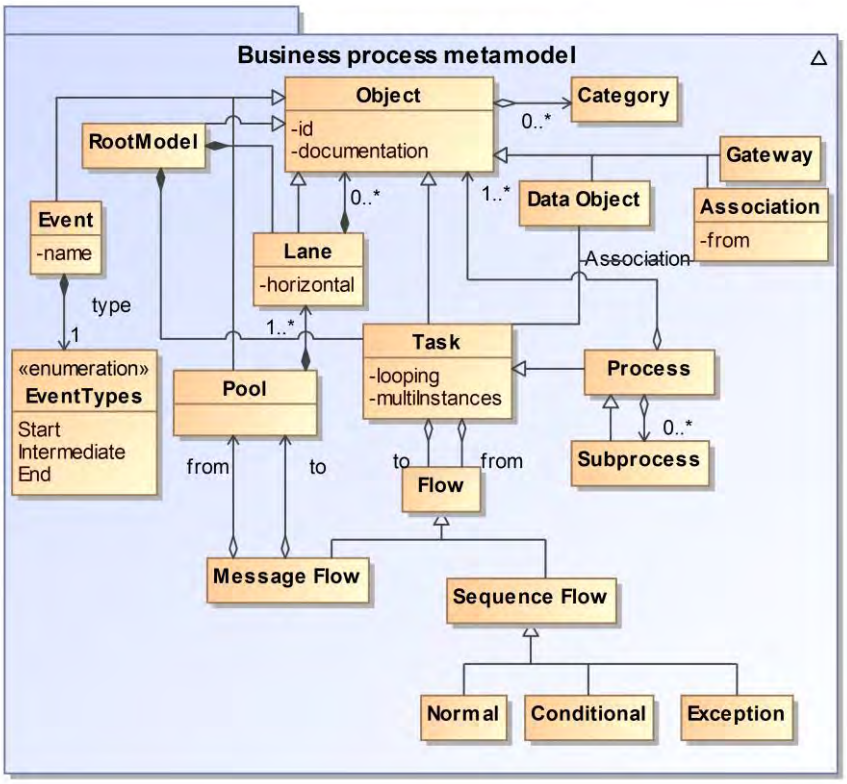


Figure IV-6: Métamodèle de processus métier

Un extrait du modèle du processus métier (cf. Figure IV-7) montre les différentes phases de gestion du CMS. Pour simplifier, nous avons limité le processus du CMS à un seul acteur (suffisant pour illustrer notre propos) : le relecteur. Quand la date de soumission est dépassée, le président invite les relecteurs à indiquer via le CMS les articles qu'ils souhaitent relire. Le relecteur déclare par la suite ses éventuels conflits d'intérêt.

Dans le processus de la Figure IV-7, l'accent est mis sur la procédure d'évaluation d'articles. Une fois le processus de relecture déclenché, le relecteur choisit un papier parmi ceux qu'il a précédemment sélectionnés. À partir de ce moment, il peut soit faire lui-même l'évaluation soit la sous-traiter à une tierce personne (à cause d'un manque de temps ou de compétences). L'évaluation est saisie ensuite dans un formulaire dans lequel il peut ajouter des commentaires.

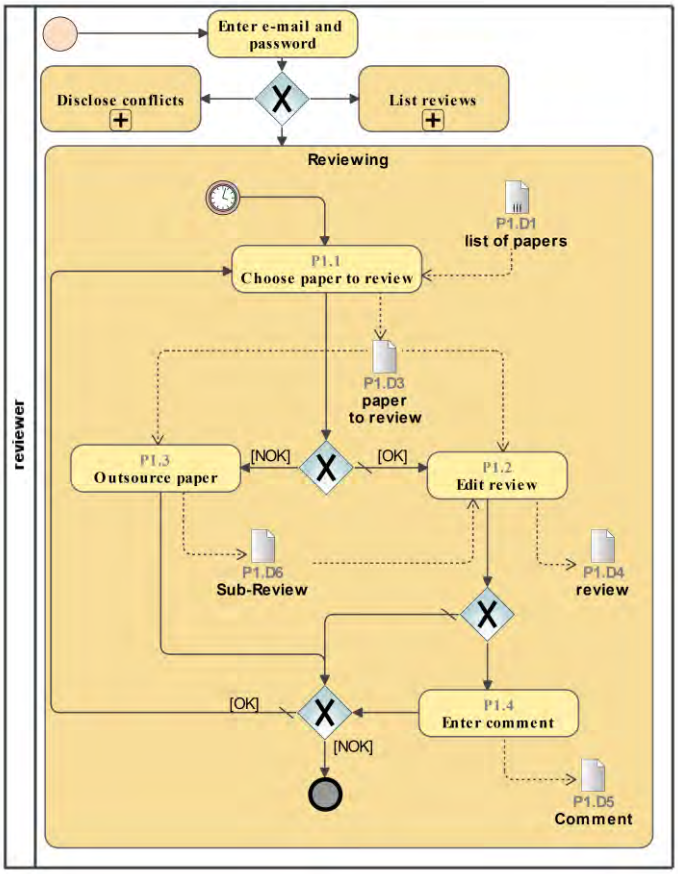


Figure IV-7: Modèle de processus métier

Les sections suivantes ont pour objectif de définir la démarche mise en œuvre pour élaborer un modèle contenant les différentes correspondances qui peuvent exister entre les modèles source en exploitant les métamodèles auxquels ils sont conformes. Elles sont illustrées par l'exemple du CMS.

IV.3. Métamodèle de correspondance

Nous proposons un métamodèle de correspondance (MMC) (cf. Figure IV-8) qui identifie les différents concepts via lesquels les modèles de correspondance sont créés. Le métamodèle introduit la notion de modèle de correspondance (*CorrespondenceModel*), qui contient les correspondances établies entre des (méta-)éléments de (méta)modèle distincts à travers leurs références. Le concept *Correspondence*, possède deux attributs *mandatory* et *weight* qui indiquent respectivement si la correspondance est obligatoire, et le coefficient de pondération qui lui est associé. L'exploitation de ces deux attributs sera traitée dans le Chapitre V. La correspondance est composée d'au minimum deux éléments référencés (*ReElement*) et du type de la relation (*Relationship*) qui les relie. Ce dernier est associé au concept *RefElement* afin de définir des correspondances de type n-aire mettant en relation plusieurs (méta-)éléments en même temps avec un type de relation. Son attribut *bidirectional* permet de spécifier si le type de relation est bidirectionnel. Si c'est le cas, les éléments concernés sont tous de type source et il n'y a aucun élément de type cible, d'où la règle OCL : *self.bidirectional implies self.targetL-> isEmpty()*. L'attribut

priority, qui est aussi exploité dans le Chapitre V, permet d'accorder à chaque type de relation une valeur de priorité. Le concept *Relationship* est une généralisation abstraite des concepts spécifiés par les concepts abstraits *DIR* (*Domain Independent Relationship*) et *DSR* (*Domain Specific Relationship*). La spécialisation du premier permet de représenter les types de relation générique, indépendantes du domaine d'application. On peut citer : l'agrégation, la similarité, la dépendance et la généralisation. Cependant ces types de relations préétablies peuvent être insuffisants. Dans ce cas il est possible de représenter des types de relations spécifiques à un domaine par spécialisation du concept *DSR*. Nous donnons des exemples de cette spécialisation dans la section IV.4 avec l'exemple du CMS, et dans l'étude de cas du Chapitre VII.

Level est un concept abstrait qui est assigné à un type de relation pour décrire son niveau. Sa spécialisation permet de distinguer deux types de niveau représentés par les concepts *HLR* (*High Level Relationship*) et *LLR* (*Low Level Relationship*). Le Concept *HLR* permet d'exprimer que le type de relation est adapté à une utilisation dans des correspondances au niveau métamodèle alors que le deuxième concept *LLR* est pour le niveau modèle. La règle OCL : *self.refined->forAll(r|r.adaptedTo->forAll(lv|lv.oclIsTypeOf(LLR) implies self.abstract.adaptedTo->forAll(lv|lv.oclIsTypeOf(HLR)))* spécifie que tout type de relation utilisable au niveau HLR est aussi utilisable au niveau *LLR*. Par contre l'inverse n'est pas vrai.

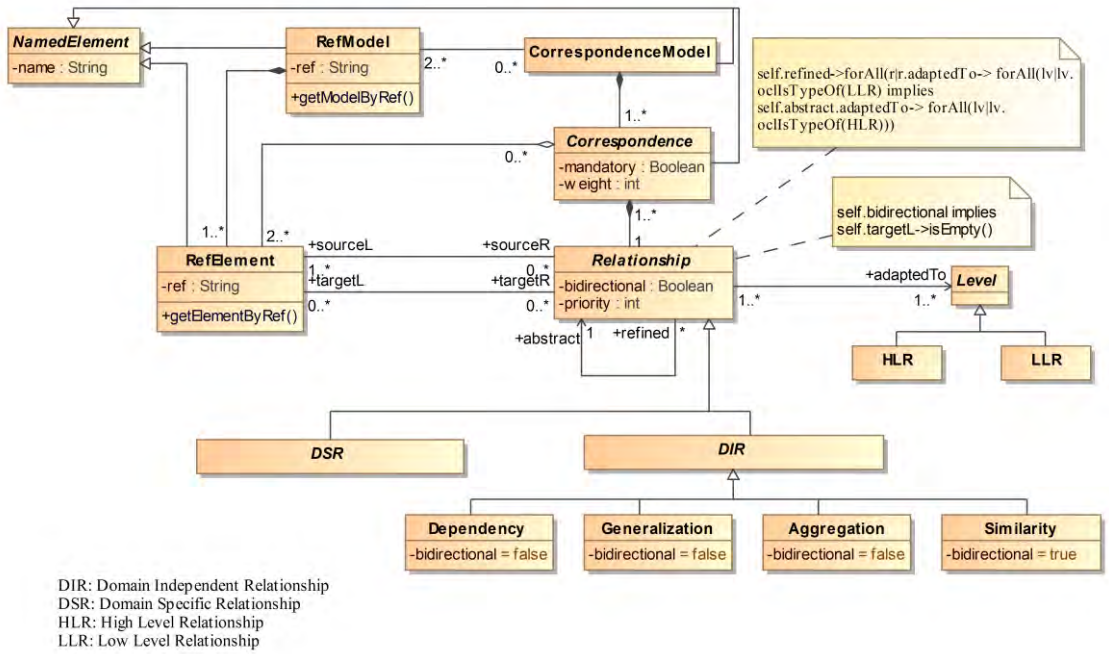
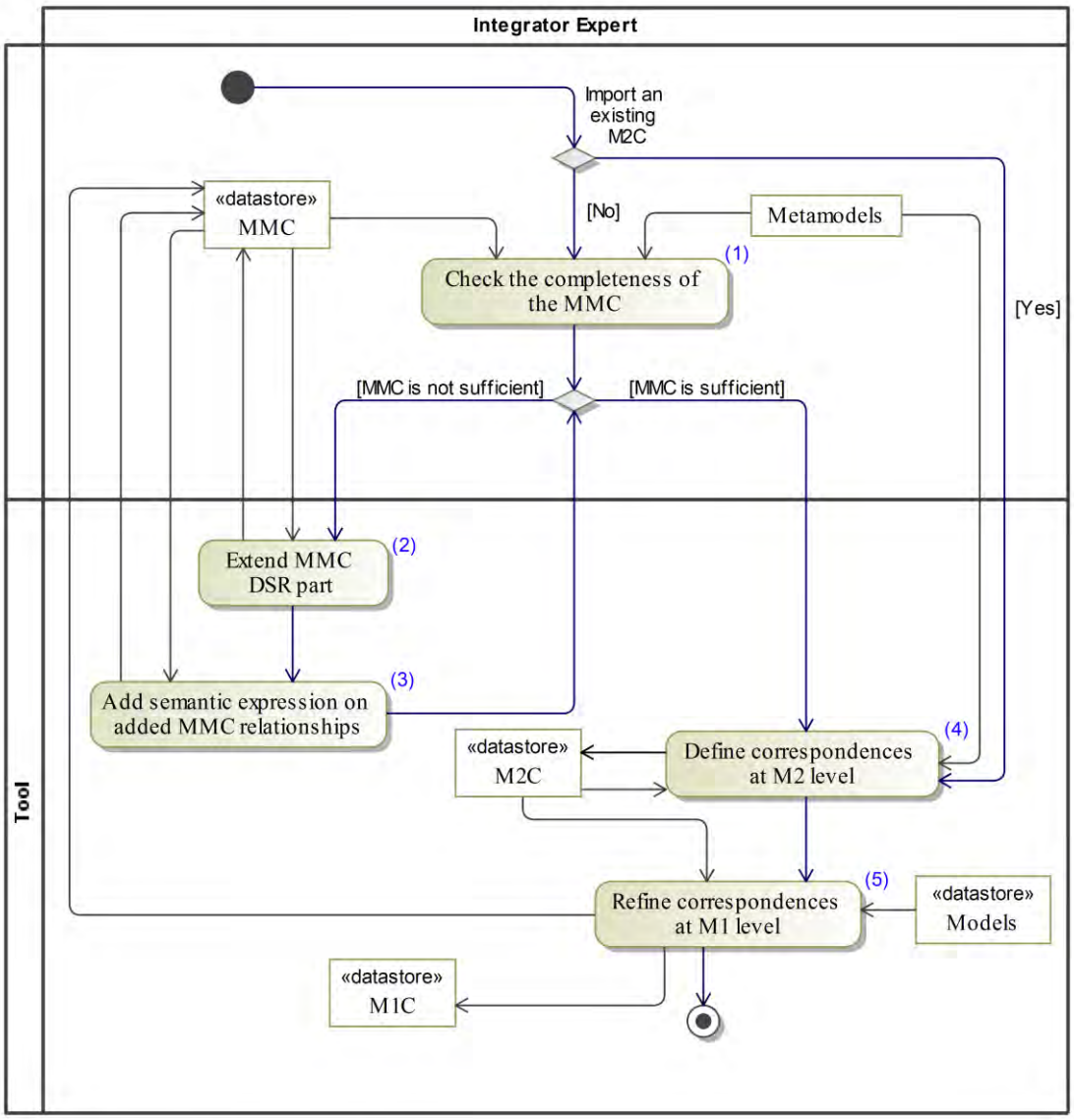


Figure IV-8 : Le noyau du métamodèle de correspondance

IV.4. Processus de Mise en correspondance

Le modèle de correspondance ne peut pas être construit de façon monolithique. Il doit être créé par l'intermédiaire d'un processus, décrit par la Figure IV-9. Ce processus a pour objectif de décrire les étapes nécessaires pour effectuer la mise en correspondance entre les modèles hétérogènes, afin d'obtenir un modèle de correspondance. Le processus est en réalité itératif mais nous ne présentons ici qu'une itération. Il implique deux acteurs. Le premier est l'*expert intégrateur* qui peut être considéré comme le chef d'orchestre des concepteurs du système étudié. Nous

posons comme hypothèse que cet acteur possède une vision globale du domaine d'application équivalente à celle du *Scrum Master* en développement agile (Malone, 2014). Cette hypothèse est réaliste car dans tout projet de développement de système complexe, il y a au moins une personne qui doit construire cette vision globale. Le second acteur, non humain, est l'*outil* qui permet d'assister l'expert intégrateur. Le niveau d'intervention de l'outil diffère d'une activité à une autre. Les activités dans lesquelles l'outil intervient avec l'expert intégrateur sont celles visibles en intersection des deux couloirs d'activité (Figure IV-9). Dans (Beugnard et al., 2014), plusieurs situations de modélisation sont présentées. Dans notre approche, nous considérons que les modèles ainsi que leurs métamodèles respectifs pré-existent et ne sont donc pas à créer pendant la mise en correspondance.



MMC : Metamodel of correspondence
DSR: Domain Specific Relationship
M2C : Model of correspondences between metamodels
M1C : Model of correspondences between models

Figure IV-9 : Processus de mise en correspondance

Le processus commence par une activité de vérification de la complétude du MMC, géré par l'expert intégrateur. Ayant en entrée les différents métamodèles et le noyau du métamodèle de correspondance (MMC), cette activité consiste à vérifier si ce dernier contient suffisamment de types de relations pour établir les correspondances entre les modèles du domaine d'application étudié. Si l'expert du domaine estime que les types de relations proposés dans le MMC ne sont pas suffisants pour exprimer les correspondances, le concept DSR (cf. Figure IV-8) est spécialisé dans la deuxième activité, pour prendre en compte les spécificités du domaine d'application, en introduisant les types de relation manquants. Dans le cas du CMS, nous avons ajouté les trois types de relations suivants : *Requirement, Contribution*, et *Play* (cf. Figure IV-10). Le premier permet de préciser les entrées nécessaires pour une tâche donnée du modèle de processus CMS. Le deuxième permet d'identifier les opérations qui contribuent au bon déroulement d'une tâche donnée. Le troisième met en évidence le rôle que jouent les participants d'une conférence dans le processus métier.

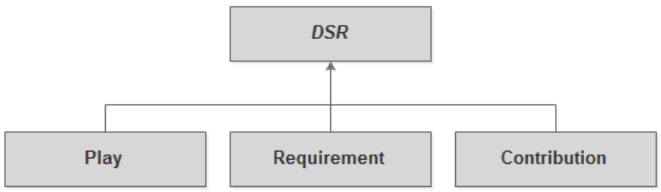


Figure IV-10 : Types de relations spécifiques pour le domaine du CMS

La troisième activité de ce processus consiste à enrichir le MMC avec une expression de la sémantique de chaque type de relation ajoutée sous forme d'annotations. Ces annotations sémantiques constituent un moyen d'enrichir la description sémantique du système complexe à modéliser. C'est l'un des apports de notre approche. La mise en œuvre de cette activité est présentée dans la section IV.5.

La quatrième activité représente la première étape de la mise en correspondance. Elle commence par l'identification des correspondances entre les méta-éléments, de manière à produire un modèle de correspondance appelé M2C. Les correspondances stockées dans le M2C sont appelées «correspondances de haut niveau (*HLC*)». Elles contiennent les méta-éléments et les *HLR* reliés entre eux. La cinquième activité permet de raffiner les *HLCs* afin de produire les correspondances de bas niveau (*LLC*) qui contiennent cette fois-ci les éléments du niveau modèle et les *LLR* reliés entre eux. Ce raffinement (*refine*), présenté avec plus de détails dans la section IV.7, étend si nécessaire le MMC en ajoutant des types de relations adaptés uniquement au niveau *LLR* (section IV.7.4) et produit par la suite le modèle de correspondance appelé M1C. Ce dernier, conforme au MMC, contient les correspondances entre les éléments de modèles.

IV.5. Ajout de l'expression de la sémantique pour chaque type de relation ajouté

Comme décrit dans la troisième activité du processus, l'expert intégrateur est amené à préciser la sémantique de chaque type de relation ajouté (de type DSR). Les types de relation de type DIR possèdent déjà une sémantique (qu'il est possible de modifier) dans la mesure où ils font partie du noyau générique du métamodèle MMC. Nous supposons que le MMC est une

entrée de notre processus de mise en correspondance donc une activité de spécification de la sémantique des types de relation a déjà été effectuée au moment de la définition du noyau générique MMC. L'activité que nous décrivons ici consiste à enrichir le MMC avec des expressions sous forme d'annotations. Dans ce qui suit nous présentons l'expression de la sémantique des types de relation de correspondance par un langage spécifique que nous appelons SED (*Semantic Expression Description*). Nous utilisons dans cette phase une sémantique opérationnelle exprimée dans un modèle conforme au SED. Ce DSL, décrit par la Figure IV-11, a été proposé afin d'attribuer et de contrôler la sémantique des différents types de relation, qui peuvent différer d'un domaine à un autre (par exemple le type de relation *Requirement* peut avoir une signification différente d'un domaine à l'autre).

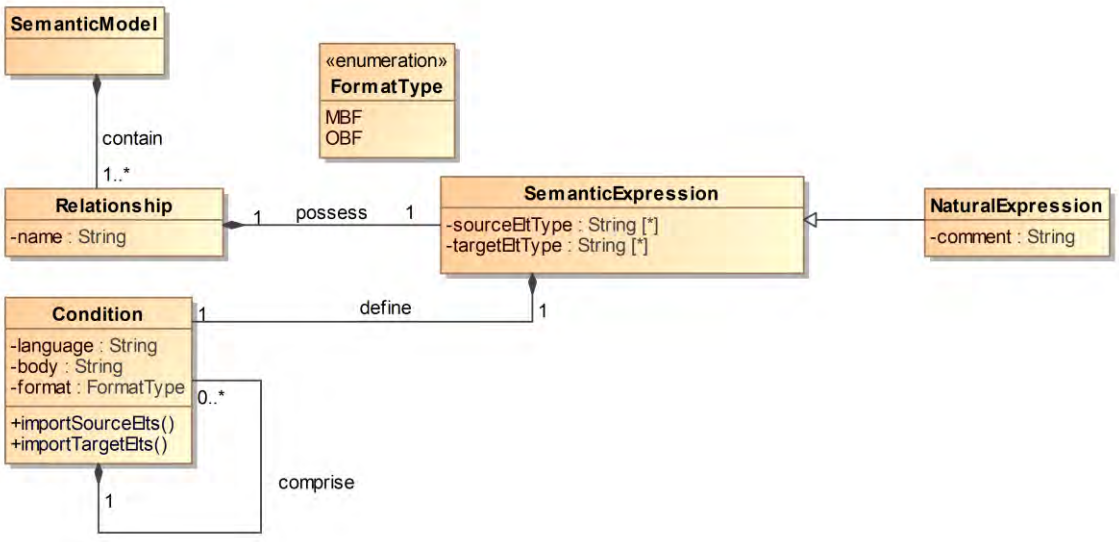


Figure IV-11 : DSL d'expression de la sémantique (SED)

L'activité d'ajout de la sémantique déclenche la création d'un modèle SE à partir du SED. Pour la création de ce modèle, un parcours du MMC doit être effectué afin de récupérer automatiquement la liste des types de relation (de base, et ceux ajoutés par l'expert intégrateur). Le SED permet à l'expert de définir pour chaque type de relation une expression de la sémantique. Chaque expression se compose du type d'élément (ou de la liste des types d'éléments) source et un type d'élément (ou de la liste des types d'éléments) cible (i.e. dans le cas du type *Similarity* on n'aura que des éléments source étant donné que ce type de relation n'est pas dirigé). L'expression permet aussi de définir une condition qui contient le corps à exécuter dans un langage qui devra être renseigné tel que : OCL, Java, HQL, Ruby, etc. L'attribut *format* permet de spécifier si le corps de la condition est défini en se basant sur des éléments de modèles (*MBF: Model Based Format*) ou bien sur des éléments d'ontologies (*OBF: Ontology Based Format*). Dans le deuxième cas, une transformation (des métamodèles du système étudié) de l'espace technologique de modèles vers celui des ontologies est nécéssaire. La condition définit aussi deux opérations permettant de récupérer les éléments source et cible d'une correspondance. Si l'expression dans un langage formel n'est pas possible, l'expert a la possibilité d'utiliser une expression (tout de même structurée) en langage naturel. Cette structuration est assurée par la définition du type des éléments à relier ainsi que la langue à travers laquelle l'expression est écrite.

La Figure IV-12 montre un extrait de modèle d'expression de la sémantique pour le système CMS. Dans ce qui suit, nous prenons l'exemple de deux types de relation indépendants

du domaine (DIR) − dont l'expression sémantique est donc récupérée du MMC − qui se différencient par le fait que le premier est décrit par un langage formel alors que le deuxième est décrit par une expression en langage naturel, et trois types de relation spécifiques au domaine (DSR). Le détail de l'expression sémantique associé à chaque type de relation est décrit dans la section IV.7.3.2. Le type de relation *Similarity,* par exemple, est utilisé dans une correspondance impliquant deux éléments (récupérés par la fonction *importSourceElts())* de types indifférents (le premier *Any* pour le premier élément et le second *Any* pour le deuxième). Il est décrit par une expression en Langage Java. L'expression explicite, à l'aide de la fonction *sameAs*, que les deux éléments sont similaires. Par ailleurs, pour le type de relation *Dependency*, étant donné qu'on n'a pas pu décrire le corps de la condition avec un langage informatique, il a été exprimé en langage naturel en précisant le type source concerné.

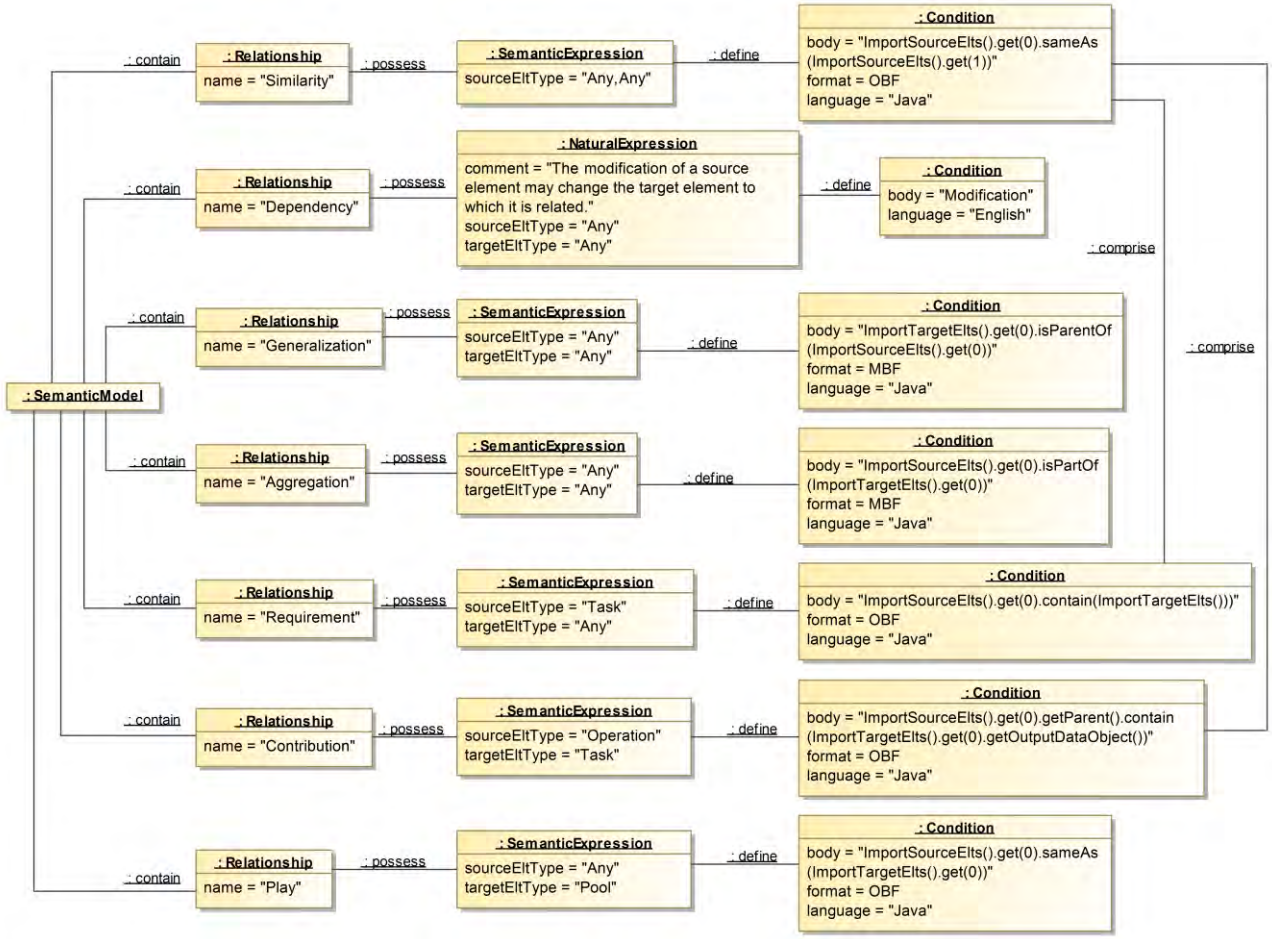


Figure IV-12 : Extrait du modèle d'expression de la sémantique pour le système CMS

Concernant le type de relation *Requirement*, de type *DSR*, il est utilisé pour relier un élément de type *Task* à un autre de type indéfini. Le corps de la condition, décrit en Java, se base sur la méthode *contain*. Concernant le type de relation *Contribution*, il repose sur la même fonction *contain* sauf que cette fois-ci elle est appliquée entre le parent de l'élément source (par exemple dans le cas du méta-élément *Operation* c'est le méta-élément *Entity,* et *Table* dans le cas de *Column*) et l'objet véhiculant les données (*DataObject*) de l'élément cible. Enfin, le corps de la condition pour du type de relation *Play* reproduit la même fonction *sameAs* sauf que cette fois-ci, l'élément cible ne peut être que de type *Pool*.

Dans cette même activité, le modèle SE d'expression de la sémantique est ensuite tissé au MMC à la manière du développement de logiciel par aspects (Filman et al., 2004) (cf. Figure IV-13). Dans notre cas, cela consiste à greffer au MMC les différentes expressions sous la forme d'annotations sur les types de relation ajoutés, comme le montre le MMC annoté résultant (Figure IV-14). Nous avons choisi l'annotation UML comme syntaxe concrète du SED.

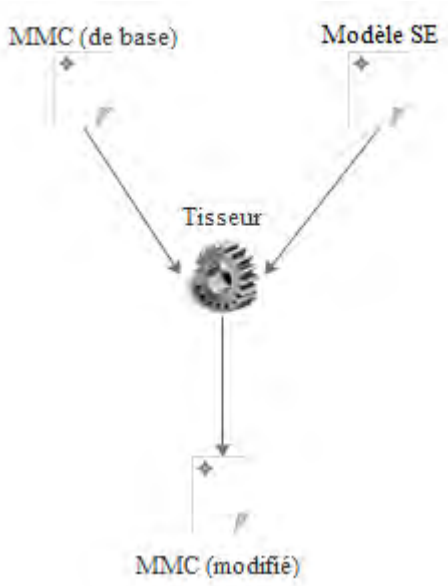


Figure IV-13 : Principe du tissage des expressions sémantiques au MMC

L'avantage de l'utilisation du DSL SED est premièrement d'avoir une définition structurée commune de chaque type de relation ; deuxièmement, cela permet de construire le modèle M2C de façon assistée à l'aide des informations sur les types d'éléments reliés (*sourceEltType* et *targetEltType*) et troisièmement cela permet d'exploiter le corps des conditions pour filtrer les correspondances lors du raffinement afin de ne garder que celles qui sont en concordance avec la sémantique du type de relation. L'exécution du corps de la condition n'a pas d'effet de bord car il est utilisé uniquement pour évaluer la validité d'une correspondance.

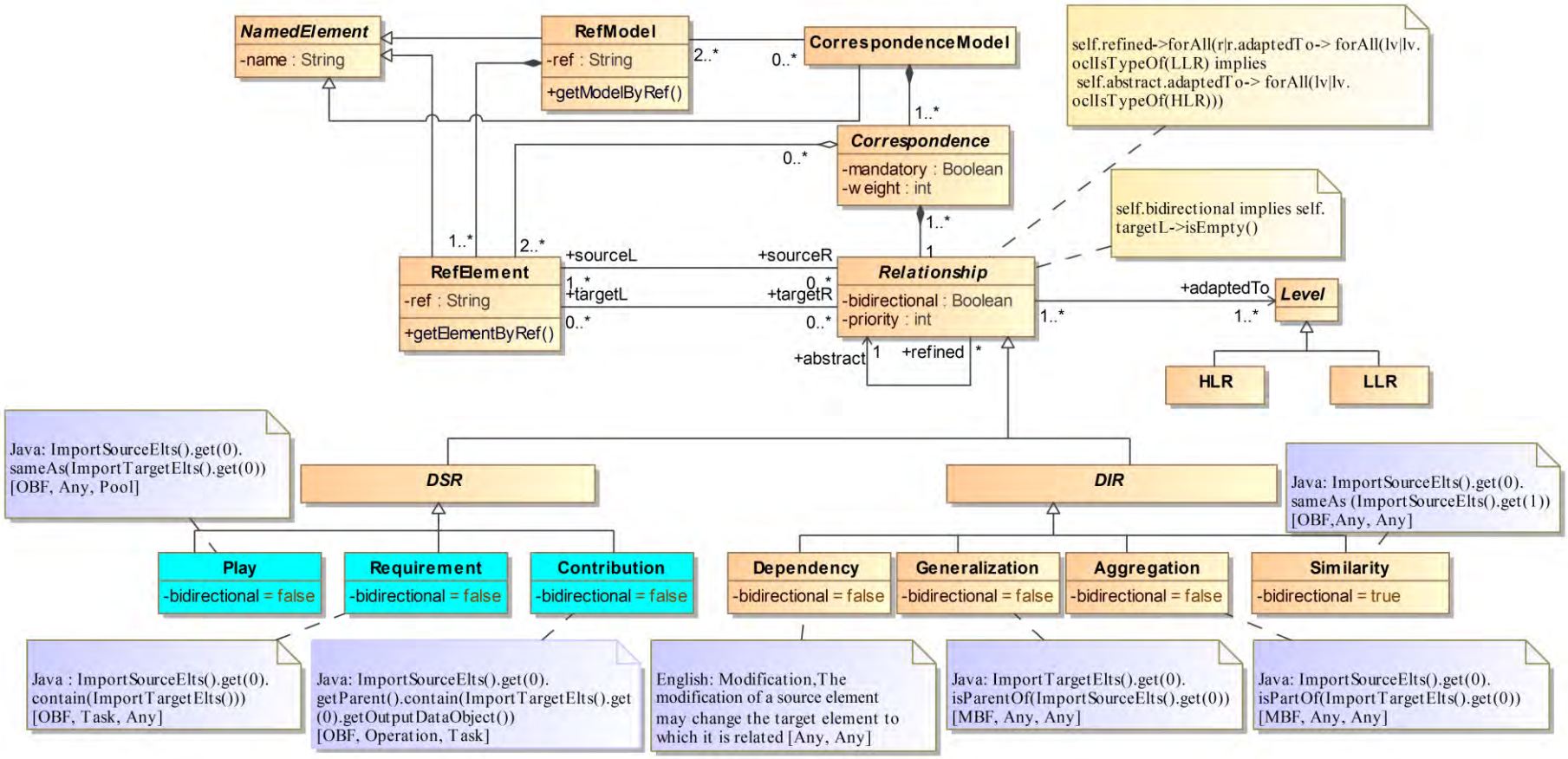


Figure IV-14 : Métamodèle de correspondance annoté pour le CMS

IV.6. Définition des correspondances au niveau M2

Les métamodèles représentent la pierre angulaire de notre approche. Les HLCs sont définies une seule fois au niveau des métamodèles du système étudié où il est plus facile d'établir les correspondances vu le nombre réduit de méta-éléments à relier comparé au nombre d'éléments au niveau modèle. L'identification d'une correspondance commence par le choix du type de relation, puis vient sa création de façon assistée par l'expert intégrateur en s'appuyant sur les annotations ajoutées précédemment au MMC conformément au SED. Plus précisément, grâce à l'information sur les types d'éléments (renseignés dans les annotations), il est possible d'appliquer un filtre sur l'ensemble des méta-éléments et de ne présenter à l'expert que les méta-éléments à relier. Par exemple, après que l'expert a choisi de représenter le type de relation *Contribution*, les méta-éléments à relier sont automatiquement précisés à partir de l'annotation attachée au type de relation. Il y a d'un côté le méta-élément *Operation* et de l'autre le méta-élément *Task*. Le type de relation *Play* par contre, ne permet d'appliquer un filtre que sur le méta-élément cible qui doit être *Pool*. Le méta-élément source doit être choisi parmi la liste des méta-éléments des métamodèles du système étudié.

La Figure IV-15 exhibe des exemples de correspondances (HLCs) établies entre les méta-éléments du CMS. Elles utilisent 5 types de relation. Par exemple, le type *Play* relie le méta-élément *StereotypedEntity* avec le méta-élément *Pool*. Le type *Contribution* créé automatiquement comme expliqué ci-dessus, relie le méta-élément *Operation* avec le méta-élément *Task*. Ce dernier est aussi utilisé dans une correspondance qui le relie, par choix de l'expert, avec *Column* par l'intermédiaire du type *Requirement,* dû au besoin dans ce cas de récupérer des éléments persistants pour éxécuter une activité. Étant donné que le type *Similarity* peut être relié avec n'importe quel élément *(Any),* il est défini manuellement entre les méta-éléments *Property* et *Column* mais aussi entre les méta-éléments *Table* et *Entity*, *Table* et *DataObject*, etc. Toutes les correspondances incluant le méta-élément *Entity* sont recréées automatiquement pour le méta-élément *stereotypedEntity*, vu que ces deux méta-éléments sont reliés par une relation d'héritage (cf. Figure IV-2). Le dernier type de relation, *Aggregation*, est utilisé pour créer la correspondance qui relie les méta-éléments *Property* et *Column*.

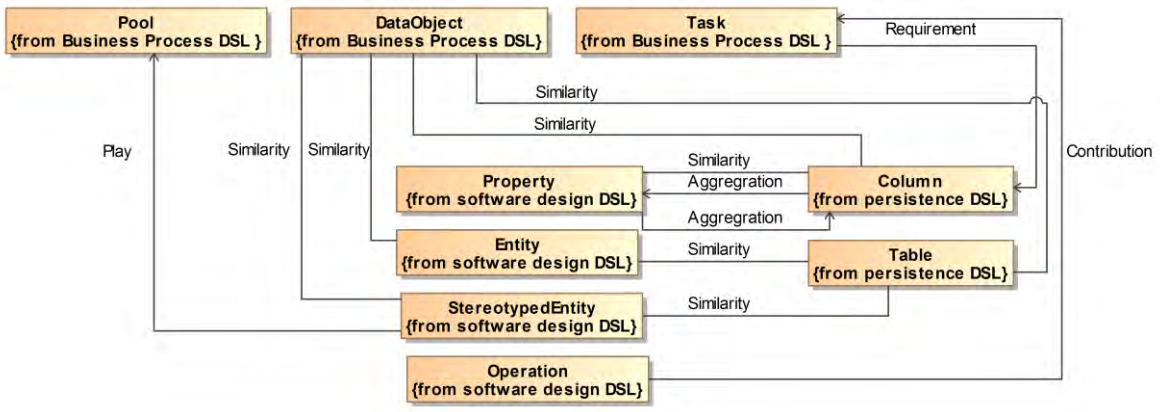


Figure IV-15 : Exemples de HLCs du CMS

À la fin de cette étape, l'expert doit attribuer à chaque type de relation une valeur de priorité (attribut *priority* dans la méta-classe *Relationship*). Les valeurs de priorité 1, 1, 2, 2, 3, 4, 5

sont attribuées respectivement aux types de relation *Similarity*, *Dependency*, *Aggregation*, *Generalization*, *Play*, *Contribution* et *Requirement*. Ces valeurs permettent de calculer des coefficients de pondération qui servent à ordonnancer le traitement des changements lors de l'évolution. L'exploitation de ces valeurs est présentée en détail dans la section V.7 du chapitre suivant.

Les HLCs servent de base contextuelle pour la création des correspondances au niveau modèle. Ce point fait l'objet des sections suivantes.

IV.7. Raffinement des correspondances au niveau M1

IV.7.1. Principe

Le principe consiste à établir une correspondance une fois au niveau métamodèle (HLC) et à la réutiliser, chaque fois que cela est nécessaire, au niveau modèle. En d'autres termes, les correspondances HLCs du niveau métamodèle induisent les LLCs du niveau modèle.

IV.7.2. Production des correspondances LLC par raffinement

Le raffinement est une manière classique de réutilisation dans l'architecture des logiciels (Moriconi et al., 1995) (Wooldridge, 1997) (Medvidovic et Taylor, 2000). Il peut être considéré comme une transition entre différents niveaux d'abstraction dans le but d'ajouter des détails lors du passage d'un niveau d'abstraction à un autre.

La notion de raffinement existe en UML. Elle est représentée comme stéréotype pour le concept *Abstraction*. Ce dernier est une relation dirigée d'un élément vers un autre indiquant que le premier (concret) dépend du second (abstrait).

Le raffinement que nous proposons peut être décrit comme une transformation endogène d'un modèle vers un autre. Elle consiste à copier certains éléments du premier modèle vers le second dans un cadre d'établissement de correspondances. Le modèle d'entrée est représenté par le M2C tandis que le modèle de sortie est représenté par le M1C. Les HLCs du M2C sont projetées sur le M1C pour donner les LLCs. Dans notre contexte, le raffinement est formalisé par la relation *refine* (notée $\boldsymbol{R_f}$) définie comme suit :

Cy $\boldsymbol{R_f}$ Cx, avec Cx et Cy deux correspondances, si et seulement si Cy est définie à partir de Cx. Ce qui signifie que Cy est plus précise que sa spécification abstraite Cx (étant donné qu'elle relie les instances des méta-éléments définis dans Cx avec la possibilité d'ajouter des détails afin de restreindre la correspondance).

On pose comme postulat que $\boldsymbol{Rf}$ est non réflexive et non symétrique. La première propiété indique qu'une HLC ne raffine pas une autre HLC alors que la seconde exprime le fait qu'une LLC est obtenue par raffinement d'une HLC et non l'inverse.

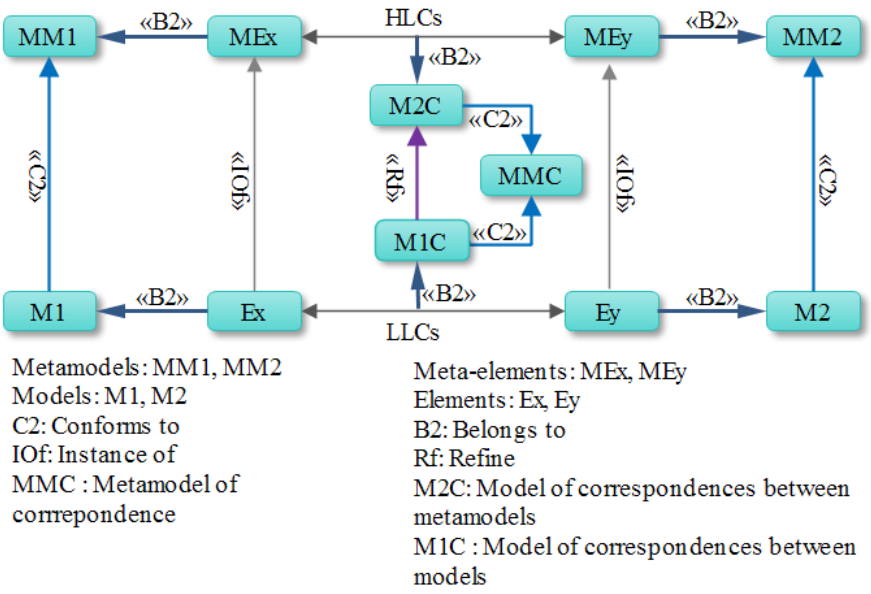


Figure IV-16 : Vue d'ensemble de la transition entre le M2C et M1C

La Figure IV-16 présente à l'aide d'un exemple abstrait les concepts de base de notre approche. Le modèle M2C contient des HLCs qui sont les correspondances reliant des méta-éléments appartenant aux différents métamodèles. Le M2C permet, par raffinement, de construire le modèle M1C, conforme aussi au MMC. Le modèle M1C contient quant à lui des LLCs qui sont les correspondances reliant des éléments appartenant aux différents modèles.

Il faut noter que dans notre approche, il est possible d'établir des correspondances de dimension supérieure à 2 entre un nombre de modèles également supérieur à 2. Dans la figure ci-dessus nous avons pris un exemple de correspondances binaire uniquement pour ne pas surcharger le schéma explicatif.

Le raffinement se déroule différemment selon que l'on apporte ou non des précisions et/ou des contraintes à la correspondance établie au niveau modèle. Dans les deux cas, la première opération consiste à réaliser une propagation des HLCs (voir section IV.7.3 ci-dessous). Si les correspondances doivent comporter des informations plus précises au regard des modèles partiels et du domaine d'application, elles seront étendues (section IV.7.4).

IV.7.3. Raffinement par propagation

Pour produire le M1C, le raffinement par propagation se déroule en deux temps. Dans un premier temps, une opération de reproduction est déclenchée sur le M2C, suivie dans un deuxième temps d'une opération de sélection. Autrement dit :

M1C = Propagation (M2C) = Sélection o Reproduction (M2C)

IV.7.3.1. Reproduction

L'opération de reproduction est un homomorphisme entre les correspondances figurant dans le M2C et celles du M1C. Il s'agit de dupliquer toutes les correspondances définies au niveau métamodèle sur le niveau modèle. En d'autres termes, il y a autant de LLCs potentielles pour une HLC donnée que le produit cartésien d'instances d'éléments pour les méta-éléments impliqués dans cette HLC.

Nous formalisons cette opération en utilisant un graphe multipartite (Jenkinson et al., 2012). Pour cela, nous définissons tout d'abord un graphe comme un ensemble de nœuds et d'arcs : G (N, A). N et A sont obtenus comme suit :

$N = N_{MM1} \cup N_{MM2} \cup N_{M1} \cup N_{M2,}$ où N_{MM1}, N_{MM2}, N_{M1}, N_{M2} sont respectivement les éléments de MM1, MM2, M1 et M2.

$(n1, n2) \in A$ SSI
- $(n1, n2) \in N_{MM1} \times N_{MM2}$ et n1 ***R*** n2, avec ***R*** un type de relation quelconque
- ou
- $(n1, n2) \in N_{M1} \times N_{MM1}$ et n1 *isInstanceOf* n2
- ou
- $(n1, n2) \in N_{M2} \times N_{MM2}$ et n2 *instantiate* n1

Étant donné que l'ensemble des nœuds est divisé en quatre sous-ensembles (MM1, MM2, M1, M2) et qu'il n'y a pas d'arc entre les éléments du même sous ensemble, cela confirme qu'il s'agit d'un graphe multipartite.

La Figure IV-17 montre les types relations (*aRx, bRz, bRy*) reliés entre les méta-éléments A, B, X, Y et Z. Nous avons orienté le graphe de telle sorte qu'il soit acyclique.

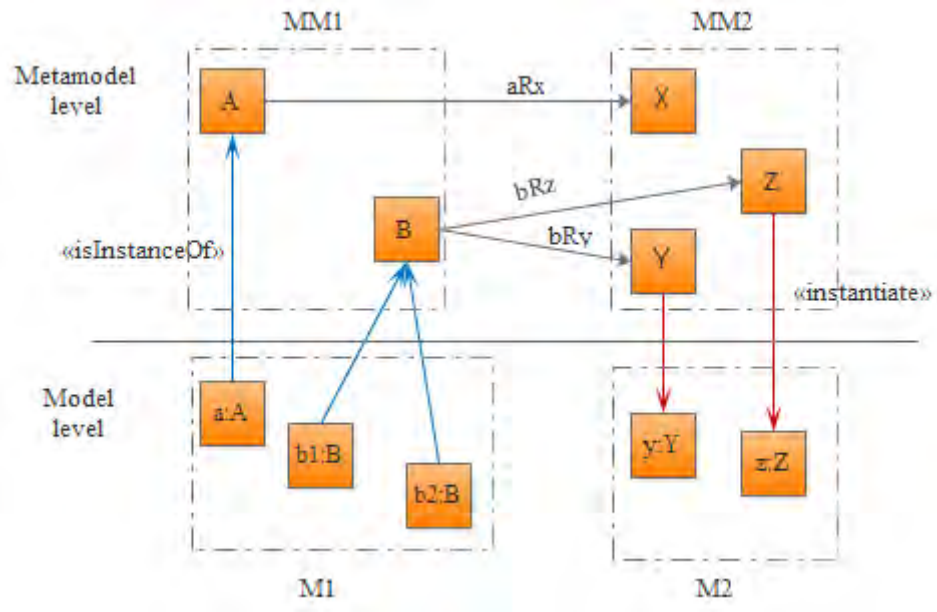


Figure IV-17 : Exemples abstraits de graphe acyclique

Pour formaliser l'opération de reproduction, la première étape consiste à produire automatiquement la matrice d'adjacence associée à chaque type de relation (de telle sorte à avoir l'équivalent de la Figure IV-17 en matrices). Les lignes et les colonnes de cette matrice correspondent aux méta-éléments et leurs éléments respectifs. L'existence d'une relation d'instanciation ou d'une correspondance est matérialisée par une valeur de 1 dans la matrice.

Pour le type *bRy* (Figure IV-17) la matrice d'adjacence est la suivante :

$$Matrice_{M2C_{bRy}} = \begin{matrix} B \\ b1:B \\ b2:B \\ Y \\ y:Y \end{matrix} \begin{bmatrix} 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Dans cette matrice, l'élément y:Y est relié à Y par une relation d'instanciation (d'où la valeur 1 à l'intersection). C'est aussi le cas pour les éléments b1:B et b2:B avec B. Par la suite, l'opération de reproduction se résume à trouver le chemin de longueur trois, de l'élément d'un modèle (exemple : b1:B) à l'élément d'un autre modèle (exemple : y:Y) en passant par le niveau métamodèle (exemple : de B et Z par la relation ayant comme type bRy) (cf. Figure IV-17).

Pour trouver ce chemin, nous calculons la matrice au cube. Cette dernière permet ainsi d'obtenir les correspondances susceptibles d'exister entre les éléments de modèles en fonction de la correspondance établie entre leurs méta-éléments. Nous pouvons observer dans la matrice ci-dessous que les éléments b1:B et b2:B sont reliés, après l'opération de reproduction, avec l'élément y:Y par le type de relation bRy.

$$Matrice_{M2C^3{}_{bRy}} = \begin{matrix} B \\ b1:B \\ b2:B \\ Y \\ y:Y \end{matrix} \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

En fusionnant la matrice au cube de chaque HLC, nous obtenons la matrice représentant l'ensemble des LLCs du M1C

$$Matrice_{M1C} = \begin{array}{c} \begin{matrix} A & a:A & B & b1:B & b2:B & X & Y & y:Y & Z & z:Z \end{matrix} \\ \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \end{array}$$

La Figure IV-18 décrit un extrait des LLCs créées par reproduction d'une des HLCs présentées dans la Figure IV-15. La HLC en question comprend le type de relation *Similarity* ainsi que les méta-éléments *Property* et *Column*. Par conséquent on aura, par reproduction au niveau modèle, des LLCs construites par produit cartésien entre les différentes instances des méta-éléments précédents.

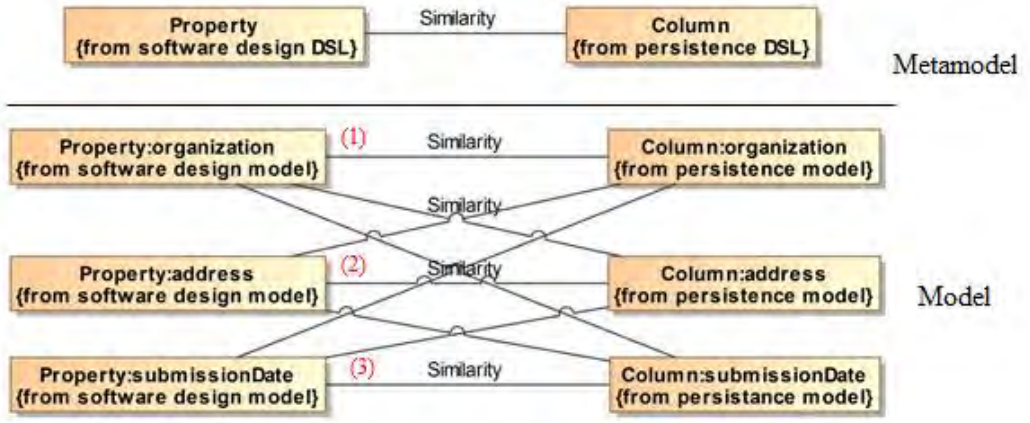


Figure IV-18: Exemple de reproduction d'une correspondance HLC avec le type de relation *Similarity*

Pour résumer, la reproduction limite la génération d'une correspondance aux éléments dont le type participe à une HLC. Cependant, même si l'information contextuelle permet la création de correspondances entre ces éléments, la reproduction ne garantit pas que toutes les correspondances générées soient sémantiquement correctes (ce qui est le cas des relations non numérotées sur la Figure IV-18). La section suivante propose une solution à ce problème via un mécanisme de sélection.

IV.7.3.2. Sélection

Cette étape consiste à filtrer les correspondances LLCs produites au cours de l'étape de reproduction, afin de ne garder que celles qui sont valides. Cela est possible grâce aux expressions sémantiques associées aux types de relation. Pour les types de relation ayant une expression informelle en langage naturel, c'est à l'expert de décider de garder ou non la correspondance en fonction de l'expression associée au type de relation. Pour les types de relation ayant une expression formelle, le corps de la condition doit être exécuté. L'exécution des corps des conditions nécessite la possession d'un interpréteur du langage avec lequel le corps de la condition a été écrit. Une machine virtuelle Java (JVM[1]) est utilisée dans notre cas.

Nous abordons dans ce qui suit les expressions formelles selon qu'elles sont décrites dans un format à base de modèle ou à base d'ontologie (attribut *format* dans le modèle SE).

IV.7.3.2.1. ***Format à base de modèle (MBF)***

Nous traitons ici la sémantique associée aux types de relation dont la valeur de leur attribut *format* est *MBF* (cf. Figure IV-12) à savoir : *Generalization* et *Aggregation*. Le corps de la condition associé à ces deux types de relation, présenté dans le modèle SE (cf. Figure IV-12), se base respectivement sur les méthodes *isParentOf* et *isPartOf*. La définition de ces méthodes s'appuie sur le thésaurus générique WordNet(Patil et Atique, 2013) via son API Java JAWS[2]. La définition de ces méthodes est la suivante :

- isParentOf :

```
public boolean isParentOf (String element)
 {
    WordNetDatabase database = WordNetDatabase.getFileInstance();
    ArrayList <String> Hyponyms = new ArrayList<String>();
    NounSynset nounS = (NounSynset)this;
    // retrieve hyponyms associated with nounS
    Hyponyms.addAll(nounS.getHyponyms());
    // check if the element in parameter exist in the list
    if (Hyponyms.contains(element)) return true;
    return false;
}
```

- isPartOf :

```
public boolean isPartOf (String element)
{
    WordNetDatabase database = WordNetDatabase.getFileInstance();
    ArrayList <String> Holonyms = new ArrayList<String>();
    NounSynset nounS = (NounSynset)this;
    // retrieve holonyms associated with nounS
    Holonyms.addAll(nounS.getHolonyms());
    // check if the element in parameter exist in the list
    if (Holonyms.contains(element)) return true;
    return false;
}
```

[1] http://java.com/fr/download/

[2] http://lyle.smu.edu/~tspell/jaws/

IV.7.3.2.2. *Format à base d'ontologie (OBF)*

Les ontologies sont une spécification formelle explicite d'une conceptualisation partagée d'un domaine d'intérêt (Gruber, 1995). Une ontologie définit les termes de base et les relations incluant le vocabulaire d'un domaine ainsi que des règles pour combiner les termes et les relations pour étendre le vocabulaire. Les ontologies sont aux bases de connaissances ce que les modèles sont au MDE. Elles définissent l'ensemble des objets du domaine de connaissances ciblé ainsi que les attributs et relations caractérisant ces objets[3](Charlet et al., 2004).

Afin de pouvoir exploiter le corps des conditions dont le format est *OBF*, on doit basculer de l'espace technologique du domaine de la modélisation à l'espace technologique du domaine des ontologies (cf. Figure IV-19). Lors de cette transformation, chaque méta-élément est représenté en un concept d'ontologie et les méta-propriétés sont représentées en des propriétés d'objets (références reliant deux objets) ou des propriétés de données (reliant un objet à une valeur de données). Ces éléments sont désignés comme des composants terminologiques (TBox). Pour les modèles, ils sont représentés sous une forme appelée *individus* d'ontologie. Chaque individu est lié à un type (méta-élément) et est décrit par un ensemble d'objets d'ontologie et de propriétés de données, définies au niveau TBox. Nous désignons ces individus comme des composants assertionnels (ABox) d'ontologies.

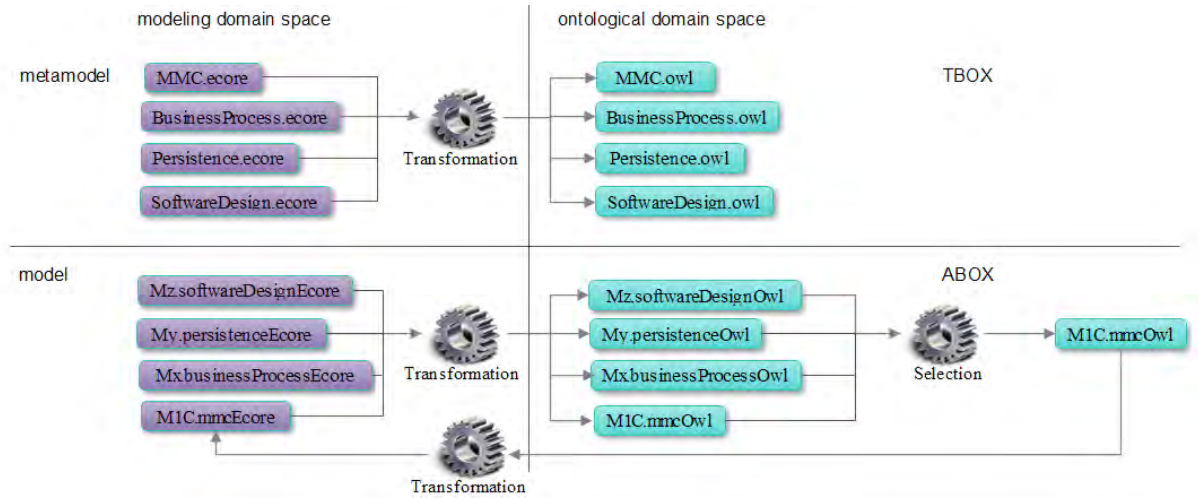


Figure IV-19: Transformation entre le domaine de la modélisation et celui des ontologies

Dans l'exemple fil conducteur, les métamodèles du CMS ainsi que le MMC sont transformés en TBox, tandis que les modèles du CMS ainsi que le M1C sont transformés en ABox.

La Figure IV-20 illustre une visualisation de TBox d'ontologie correspondant au métamodèle de persistance de la Figure IV-4.

[3] Les objets ne sont pas pris dans un sens informatique mais comme objets du monde réel que le système modélise

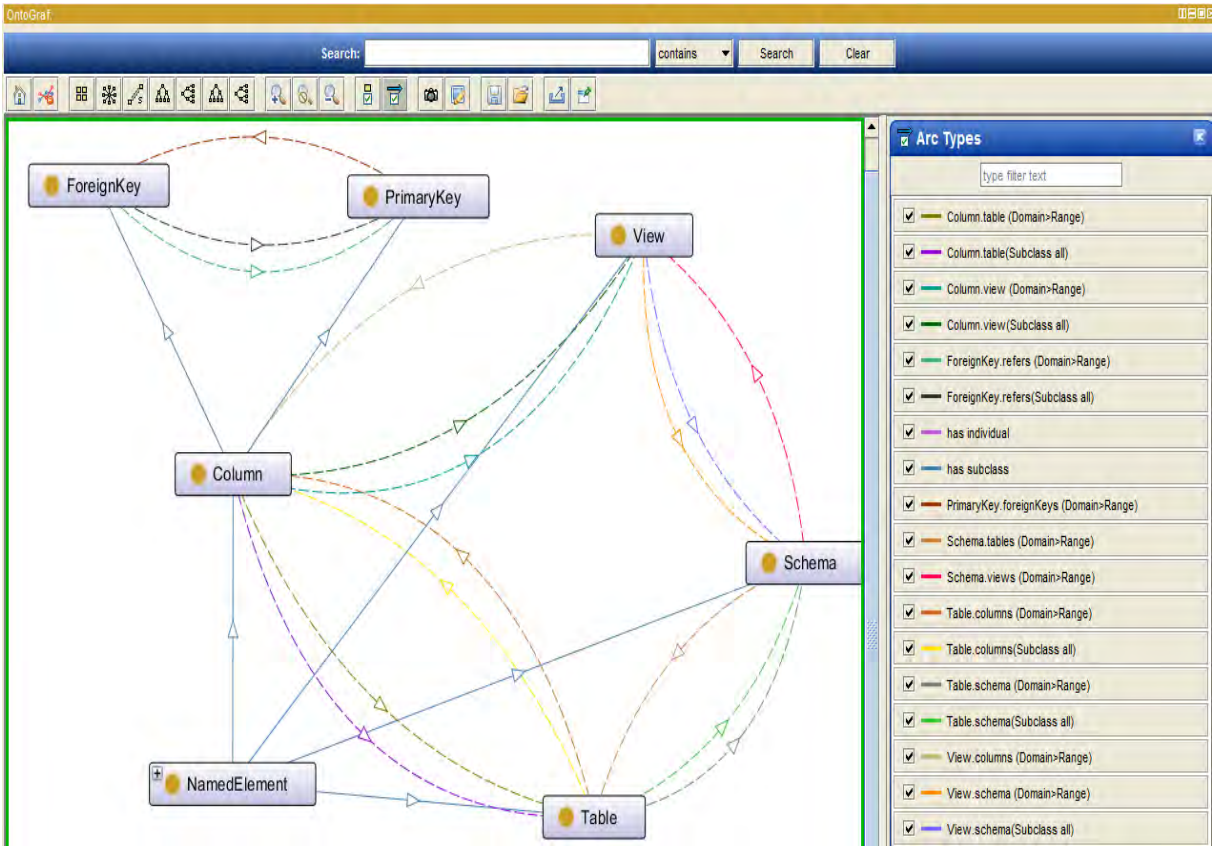


Figure IV-20 : Ontologie correspondant au métamodèle de persistance du système CMS sous Protégé[4]

L'étape qui suit consiste à exécuter le corps de la condition associé à chaque expression. Les types de relation *Similarity* et *Play* définissent dans le corps de leurs conditions une fonction *sameAs*. Cette fonction, décrite à la fin de cette section, récupère l'information à partir d'une ontologie de domaine en utilisant l'API OWL API[5]. Si cette dernière n'existe pas ou si les éléments recherchés n'y sont pas, la fonction fait appel à un thésaurus générique. Dans notre cas nous avons choisi d'utiliser l'API Java WS4J[6] de wordNet. Le calcul de similarité se fait de façon lexicographique en utilisant une mesure basée sur le contenu de l'information décrite par Lin (Lin, 1998). D'autres mesures peuvent être appliquées comme HSO (Hirst et St-Onge, 1998), (Leacock et Chodorow, 1998), LESK (Banerjee et Pedersen, 2002), etc. Si aucun résultat n'est trouvé, la fonction *sameAs* utilise, en dernier recours, une implémentation de la méthode *levenshteinDistance* définie dans une API d'alignement d'ontologies[7]. À noter que l'utilisation de cette métrique est possible car son implémentation est déjà disponible mais rien n'empêche d'en utiliser d'autres. Un panorama des métriques structurelles est présenté dans (Cheatham et Hitzler, 2013). Concernant les types de relation *Contribution* et *Requirement*, le corps de leurs conditions se base sur la fonction *contain*. Elle consiste tout d'abord à découper une phrase en plusieurs mots (via le principe de *tokenization* (Cheatham et Hitzler, 2013)) puis d'y appliquer la fonction *sameAs*.

[4] http://protege.stanford.edu/

[5] http://owlapi.sourceforge.net/

[6] https://code.google.com/p/ws4j/

[7] http://alignapi.gforge.inria.fr/lib.html

Les fonctions sameAs et contain (présentées ci-dessus) sont définies comme suit :

- sameAs :

```
public boolean sameAs(String element)
{
    OWLOntologyManager manager = OWLManager.createOWLOntologyManager();
    // load the ontology to be imported
    OWLOntology kb = manager.loadOntologyFromOntologyDocument(new
    StringDocumentSource(kb_owl));
    boolean state = false;
     // Look up for equivalent of element or current object in all subclasses
    for (OWLClass c : kb.getClassesInSignature()) {
    if(c.getName().equals(this)||c.getName().equals(element))
            if(c.getName().isEquivalentTo(this)||c.getName().isEquivalentTo(element))
            state = true
}
    if (state) return true;
    /* if the data is not found in the ontology or if there is no ontology to be imported, use lexical
    comparison */
    else{
    ILexicalDatabase db = new NictWordNet();
    //Apply Lin method
    RelatednessCalculator rc = new Lin(db);
    double score = rc.calcRelatednessOfWords(this, element);
    if (rc > 0.7) return true;
    /* if the data is not found lexicographically, use structural comparison
    else {
            uri1 = new URI( file:ontology1.owl );
            uri2 = new URI( file:ontology2.owl );
             AlignmentProcess al = new StringDistAlignment();
              al.init( uri1, uri2 );
    //Apply levenshteinDistance method
             params.setProperty("stringFunction","levenshteinDistance");
            al.align( (Alignment)null, params );
             for (Enumeration e = al.getElements(); e.hasMoreElements();)
                {
                     Cell cell = (Cell)e.nextElement();
                    String ent1 = cell.getEntity1();
                    String ent2 = cell.getEntity2();
                    if (ent1.equals(this)&& ent2.equals(element)&&cell.getMeasure > 0,7)
                    return true;
                }
        }
    return false;
}
```

- contain :

```
public boolean contain (String sentence)
{
// split the sentence parameter in several words
String [ ] tokens = tokenizer.tokenize (sentence);
//Look up for similarity between the tokens and the current object
for (int i = 0; i < tokens.length; ++i)
if (this.sameAs (tokens[i])) return true
return false;
}
```

Les fonctions *importSourceElts( )* et *importTargetElts( )* récupèrent respectivement la valeur de la propriété *name* (i.e., *organization*, *address*, *submissionDate* sont les valeurs de la propriété *name* des éléments instances du méta-élément *Property*) des éléments source et cible.

La Figure IV-21 illustre le modèle M1C obtenu à l'issu de l'étape de sélection. Par exemple, l'exécution de la fonction *sameAs* suivante : "Author".sameAs ("AuthorTable") retourne vrai étant donné que les deux éléments sont structurellement similaires, d'où le maintien de la correspondance concernant ces deux éléments. De même, l'exécution de ces deux portions de code : "fisrtName".isPartOf("fullName") et "lastName".isPartOf("fullName") retourne vrai d'où le maintien de la correspondance avec le type de relation *Aggregation* entre les éléments source (*firstName*, *lastName*) et l'élément cible (*fullName*).

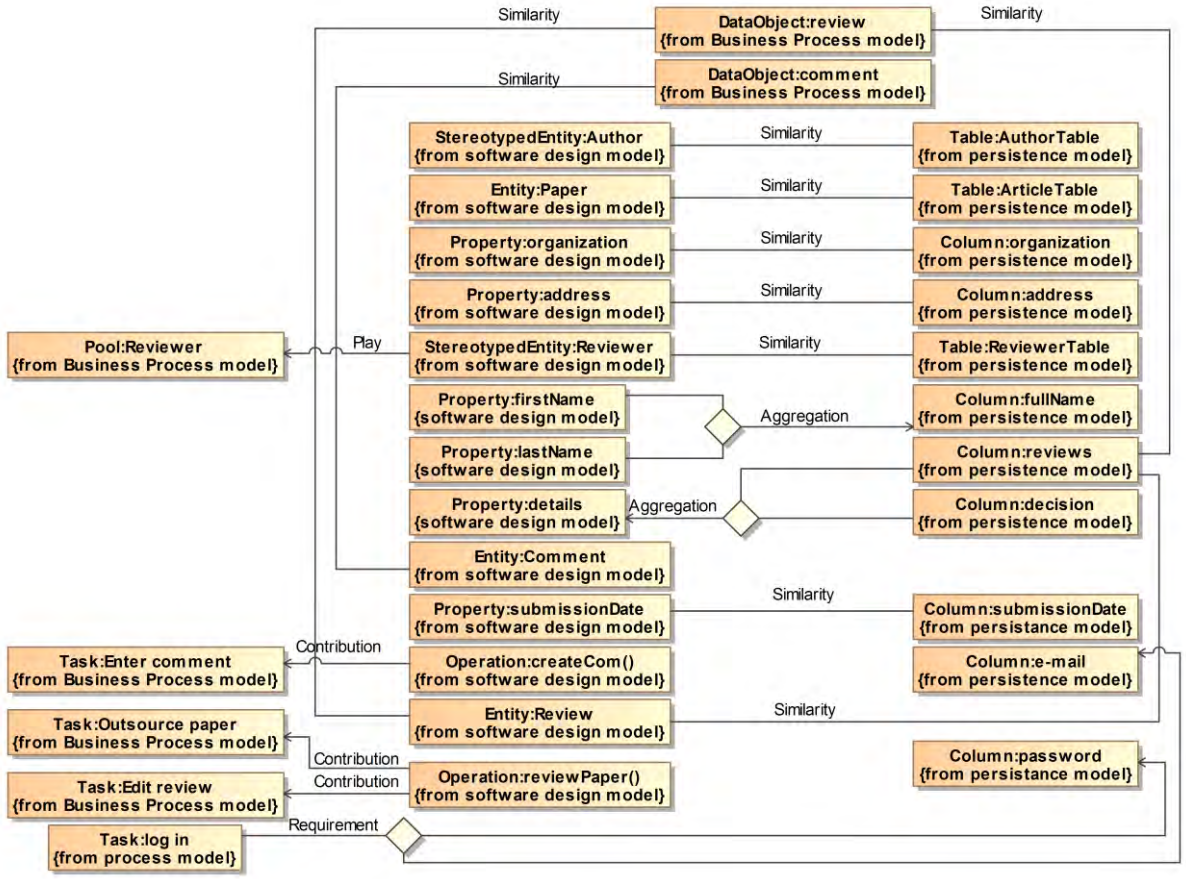


Figure IV-21 : LLCs du CMS obtenues par raffinement en propagation

Contrairement à la phase de reproduction, la validité des LLCs issue de la phase de sélection ne dépend pas uniquement du type des méta-éléments reliés mais aussi de la sémantique associée aux types de relations.

IV.7.4. Raffinement par extension

Le raffinement permet aussi de compléter, en cas de besoin, la description des LLCs avec les fonctionnalités requises pour renseigner précisément le type de relation utilisé dans la correspondance. En effet, les LLCs créées par reproduction ne satisfont pas forcément les

souhaits de l'expert. Il peut avoir à opérer des choix sur certaines actions à effectuer (Cariou et al., 2009) afin de préserver les propriétés souhaitées, ou pour ajouter des détails ou des informations sur les correspondances.

Le raffinement par extension, permet de redéfinir les correspondances afin d'ajouter aux types de relation des contraintes et/ou des traitements spécifiques du domaine d'application traité. Dans ce cas, l'extension prend la forme d'un nouveau type de relation. Ce dernier est de type LLR et hérite du type de relation qui a permis son extension.

Dans le CMS, on peut considérer que certaines correspondances ayant le type de relation *similarity* ou *aggregation* ne sont pas suffisantes ou qu'elles possèdent une sémantique vague par rapport au domaine d'application. Elles seront donc remplacées par raffinement par d'autres types de relation (cf. Figure IV-22). Le premier type (*Similarity)* est remplacé par le type *Equality* afin de décrire le fait que les éléments reliés représentent deux facettes différentes d'une même entité. (i.e. deux éléments sont reliés par le type de relation *Equality* s'ils contiennent la même valeur et que leurs conteneurs sont similaires. Exemple : *Paper* est le conteneur de la propriété *submisionDate* et d'autres propriétés dans le modèle de conception logicielle du CMS). Selon le même principe, les éléments *organization*, *address* et *submissionDate* du modèle de conception logicielle sont reliés avec les éléments du même nom dans le modèle de persistance. Le second type de relation (*aggregation*) est remplacé par le type de relation composition. Il est utilisé pour exprimer une forme stricte d'agrégation, où les cycles de vie des éléments (les composants) et de l'agrégat sont liés. Ainsi, sur la Figure IV-22, on voit que l'élément *fullName* du modèle de persistance est composé des éléments *firstName* et *lastName* du modèle de conception logicielle.

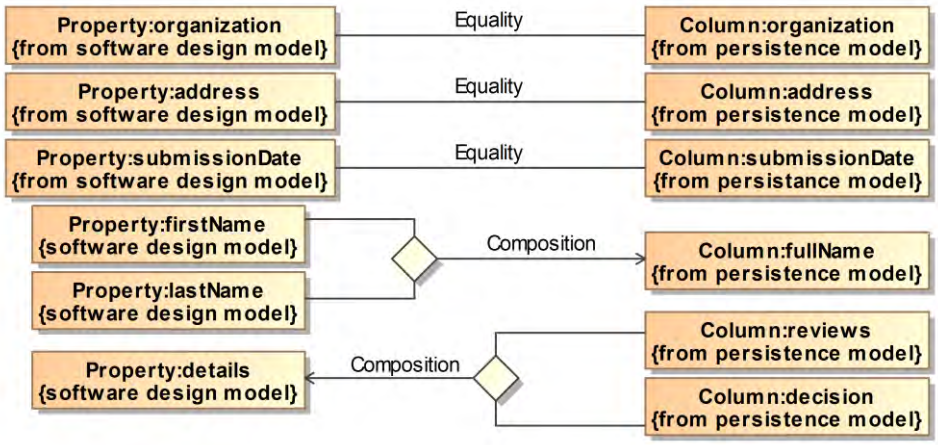


Figure IV-22 : LLCs du CMS obtenues par raffinement d'extension

IV.8. Evaluation

Cette évaluation a été réalisée en utilisant l'exemple du CMS. Pour simplifier, nous nous sommes concentrés sur le type de relation *Similarity*. Les correspondances au niveau métamodèle (HLCs) ont été créées semi-automatiquement (cf. section IV.6) ; par contre celles au niveau modèle (LLCs) ont été créées manuellement pour servir de référence d'évaluation de notre approche (sauvegardé dans un modèle de référence). Pour l'évaluation, nous avons appliqué des mesures bien connues dans le domaine de la recherche d'information telles que la précision (*Precision)*, le rappel (*Recall)* et *F-Mesure (F-measure)*. Chacune de ces métriques retourne une valeur

entre 0 et 1. La précision est le rapport entre les correspondances correctes et le nombre total de correspondances trouvées. Une valeur de précision élevée signifie que les correspondances trouvées ont plus de chance d'être correctes. Le rappel est le rapport entre les correspondances correctes et le nombre total de correspondances attendues (le nombre de correspondances dans le modèle de référence). Plus la valeur de rappel est élevée moins le nombre de correspondances non retrouvées est faible. F-mesure reflète la moyenne harmonique de la précision et du rappel. Elle est calculée da la façon suivante : F-mesure=2 * (Précision * Rappel) / (Précision + Rappel). Dans ce qui suit, nous considérons les correspondances entre les paires de modèles sachant qu'il est possible dans notre approche d'établir les correspondances entre plus de deux modèles. Nous considérons aussi que la sémantique d'expression du type *Similarity* est définie en utilisant uniquement la méthode *levenshteinDistance*.

Le Tableau IV-1 présente les résultats de notre évaluation. Les deux opérations de raffinement (reproduction et sélection) ont été effectuées indépendamment les unes des autres, par rapport au type de relation considéré. Comme le montre la Figure IV-15, plusieurs HLCs sont établies impliquant sept types de relation parmi lesquels le type de *Similarity*. À travers l'opération de reproduction (cf. section IV.7.3.1), toutes les LLCs possibles sont générées. La première colonne représente le résultat de l'application de cette opération. La deuxième opération de sélection consiste à filtrer les correspondances incorrectes générées à l'issu de l'opération précédente. Le résultat de cette évaluation est représenté dans la seconde colonne du tableau.

Pour la paire de modèles *SoftwareDesign-Persistence* (SD-PS), 462 correspondances ont été créées suite à l'opération de reproduction. Comme prévu, la précision est très faible (dû au fait que beaucoup de correspondances incorrectes ont été générées) tandis que le rappel est élevé (dû au fait que beaucoup de correspondances correctes ont été trouvées). Pour la même paire de modèles, de meilleurs résultats sont obtenus avec l'opération de sélection (P= 0.65, R = 0.45 et F = 0.52). Pour la paire de modèles *BusinessProcess-SoftwareDesign* (BP-SD), bien que la reproduction génère le meilleur rappel, la sélection produit une meilleure performance en termes de précision (P = 0.66) et de F-mesure (F=0.66). La réduction de la valeur du rappel dans la séléction est justifiée par l'élimination d'un ensemble de correspondances lors de la sélection avec un filtre qui se base sur la méthode *levenshteinDistance*. Enfin, pour la paire de modèles *BusinessProcess-Persistence*, un comportement inverse a été observé où les valeurs des mesures sont à 0. Ceci est dû aussi à l'expression de la sémantique définie pour ce type de relation et de ce fait, aucune correspondance, par rapport au modèle de référence, n'a pu être retrouvée.

[1]Précision [2]F-mesure [3]Rappel [4]*SoftwareDesign* [5]*Persistence* [6]*BusinessProcess*

	Reproduction (*Similarity*)			**Sélection (Reproduction (*Similarity*))**		
	P[1]	F[2]	R[3]	P	F	R
SD[4]-PS[5]	0.01	0.02	0.63	0.65	0.52	0.45
BP[6]-SD	0.06	0.12	1.00	0.66	0.66	0.66
BP-PS	0.01	0.03	1.00	0.00	0.00	0.00

Tableau IV-1 : Résultats de l'évaluation de la sélection

Lors de cette évaluation, nous nous sommes appuyés sur la sémantique structurelle en utilisant uniquement la méthode *levenshteinDistance*. Nous sommes persuadés qu'en exécutant des codes détaillés de l'expression de la sémantique tels que définis dans la section IV.7.3.2, de

meilleurs résultats pourraient être obtenus. Ce travail n'a pas encore été fait étant donné que le SED et l'opération de sélection ne sont pas totalement implémentés dans notre outil.

IV.9. Conclusion

Dans ce chapitre, nous avons décrit l'approche proposée pour la mise en correspondance de modèles hétérogènes. Nous avons proposé une formalisation de notre démarche sous forme d'un processus décrit selon le standard UML (diagramme d'activité). Notre démarche s'appuie en premier lieu sur un métamodèle de mise en correspondance qui présente plusieurs avantages :

- Généricité : le métamodèle de correspondance (MMC) fournit une partie «générique», commune à tous les domaines, qui permet de décrire la plupart des types de relation courants. Pour cela que par exemple, nous avons utilisé la *similarity* présente dans la majorité des approches étudiées et familière aux utilisateurs d'UML,
- Extensibilité : Le MMC peut être étendu en fonction des particularités du domaine d'application considéré, afin de gérer les types de relations propres à des domaines métiers spécifiques. Cela se fait par des spécialisations du concept *DSR* du MMC,
- Flexibilité : Le MMC permet de relier n modèles à la fois (à travers leurs éléments de modèle) et d'exprimer des correspondances n-aire (n>=2),
- Légèreté : Le MMC permet de créer un modèle de correspondance construit de manière virtuelle (Clasen et al., 2011b, Clasen et al., 2011a) dans la mesure où ce modèle ne contient que des références aux éléments physiques.

En deuxième lieu, afin notamment de minimiser le travail de l'expert lors du raffinement, nous avons fourni un moyen de description de la sémantique des types de relation à travers le langage spécifique SED. En dernier lieu, nous avons décrit le mécanisme de raffinement qui permet de produire semi-automatiquement le modèle de correspondance. Contrairement aux approches étudiées dans le Chapitre III, un seul modèle est créé regroupant l'ensemble des correspondances existant entre les différents modèles du système à développer.

Le modèle de correspondance produit peut être utilisé pour différentes finalités. Il permet tout d'abord d'avoir une vue globale du système (cf. Figure IV-1). Chaque acteur possède non seulement une vue sur son modèle partiel mais aussi une vue plus globale à travers le modèle de correspondance. Ce dernier utilise le mécanisme de virtualisation pour accéder aux éléments de modèles des différentes correspondances, ce qui le promeut au rang de modèle global virtuel. Ce modèle peut être utilisé aussi dans un contexte d'interopérabilité, permettant ainsi aux différents acteurs d'échanger des informations et de les exploiter. Grâce au modèle de correspondance, en utilisant un Framework de transformation d'un modèle vers du texte (M2T) et en précisant la plateforme technique cible, il serait possible de générer partiellement du code afin de construire par exemple le schéma de la base de données du système, du code applicatif à exécuter, etc. L'un des axes où le modèle de correspondance produit s'avère important est le maintien de la cohérence des modèles suite à une évolution. Ce point est specifiquement traité dans le chapitre suivant.

CHAPITRE V. MAINTIEN DE LA COHÉRENCE LORS DES ÉVOLUTIONS DE MODÈLES

"Each type of model requires a particular set of skills to produce and to evolve effectively. (…) the interrelationships between multiple types of models, and potentially, different modeling formalisms, suggests that it will be difficult for any given stakeholder (for example: use case developer, architect, implementor, tester) to understand the impact of a proposed change on all the related artifacts"

— *Brent Hailpern, Peri Tarr*

V.1. Introduction

Les modèles partiels que nous considérons dans cette thèse sont toujours évolutifs pendant la phase de conception/modélisation voire même pendant les phases aval du développement du système, et à l'image de leur élaboration qui a été faite séparément par les différents acteurs (concepteurs), leur évolution peut se produire de façon non coordonnée. Notre objectif est donc de décrire comment on peut assurer la synchronisation de ces modèles partiels c'est-à-dire maintenir la cohérence du réseau de modèles constitué en exploitant le modèle de correspondance M1C dont nous avons présenté la construction dans le chapitre précédent.

Une démarche possible consiste à relancer le processus de mise en correspondance (Figure IV-9) lors des évolutions de modèles. Mais cette solution ignore le travail fait en amont en reconstruisant le modèle de correspondance «*from scratch*». Par ailleurs, aucune trace sur les changements effectués ne sera répertoriée. Cette démarche, non optimale, est celle adoptée par la majorité des approches de composition de modèles étudiées.

Dans notre travail, nous exploitons le M1C pour assurer la gestion de la cohérence dans le cas où les modèles partiels évoluent. En effet, lorsque le processus de mise en correspondance a produit le M1C, le processus de maintien de la cohérence est activé (cf. Figure V-1). Il prend en entrée l'ensemble des modèles partiels du domaine d'application et le modèle de correspondance M1C.

Il faut noter que dans notre approche, mis à part les éléments ajoutés, on ne traite que les changements (de type modification et suppression) qui ont une répercussion sur les éléments figurant dans le modèle de correspondance M1C. En d'autres termes, la gestion de la cohérence concerne les éléments de modèles qui participent à une correspondance. Elle ne traite pas l'incidence d'une modification de modèle sur le même modèle (à moins qu'il existe une

correspondance réflexive qui concerne deux éléments appartenant au même modèle dans le M1C). C'est aux concepteurs des différents modèles source de gérer la répercussion en interne des changements faits sur leurs modèles et de s'assurer de leurs validités.

Le processus, que nous décrivons dans la section V.2, comprend six phases majeures : les deux premières phases concernent l'identification des changements et leur analyse. Elles font l'objet des sections V.3 et V.4. La gestion des cycles est traitée dans la section V.5. La section V.6 aborde les stratégies d'ordonnancement des changements tandis que la section V.7 traite de leur priorisation. La section V.8 est dédiée aux traitements des répercussions des changements sur les modèles du domaine d'application. Les différentes phases de ce processus sont illustrées sur notre exemple fil conducteur CMS.

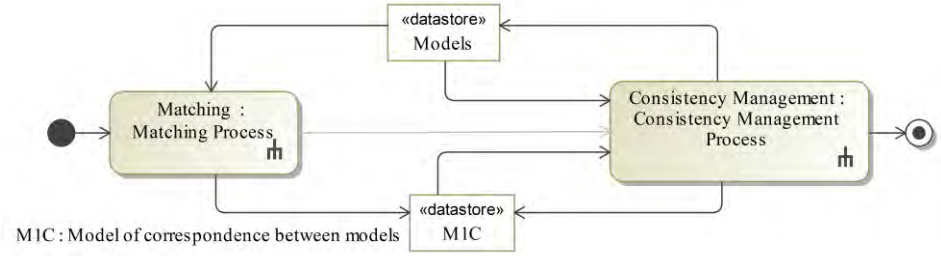


Figure V-1 : Processus global de notre approche

V.2. Processus de maintien de la cohérence

Notre objectif est de proposer une gestion de la cohérence des modèles via le modèle de correspondance construit selon la démarche présentée au Chapitre IV. Nous proposons pour cela un processus en six phases (décrites sur la Figure V-2). Pour commencer, la phase de détection des changements a pour objectif de garder la trace des changements qui se sont produits sur les modèles. Ces changements sont stockés dans le M1C. Ensuite, après décision de l'expert, une analyse automatique des changements est effectuée. Elle consiste à renseigner, pour chaque changement, son mode de classification et les répercussions directes et indirectes susceptibles d'avoir lieu. La phase qui suit a pour objectif d'éliminer les cycles afin d'automatiser le traitement des changements. Par la suite, une première liste des changements est produite après que l'expert a choisi la stratégie à adopter. Cette liste est affinée lors de la quatrième phase de priorisation par le calcul de coefficients de pondération. Ces derniers servent à déterminer l'ordre de traitement des changements. Enfin, la phase de traitement des changements applique des actions spécifiques pour chaque changement. Certaines d'entre elles s'effectuent de façon automatique alors que d'autres nécessitent l'intervention de l'expert et des concepteurs des modèles concernés par l'évolution. Dans le cas où d'éventuels changements ont été détectés suite à cette dernière phase, le processus de matien de la cohérence est relancé. À la fin de ce processus le modèle de correspondance actualisé reste cohérent vis-à-vis des modèles du domaine d'application.

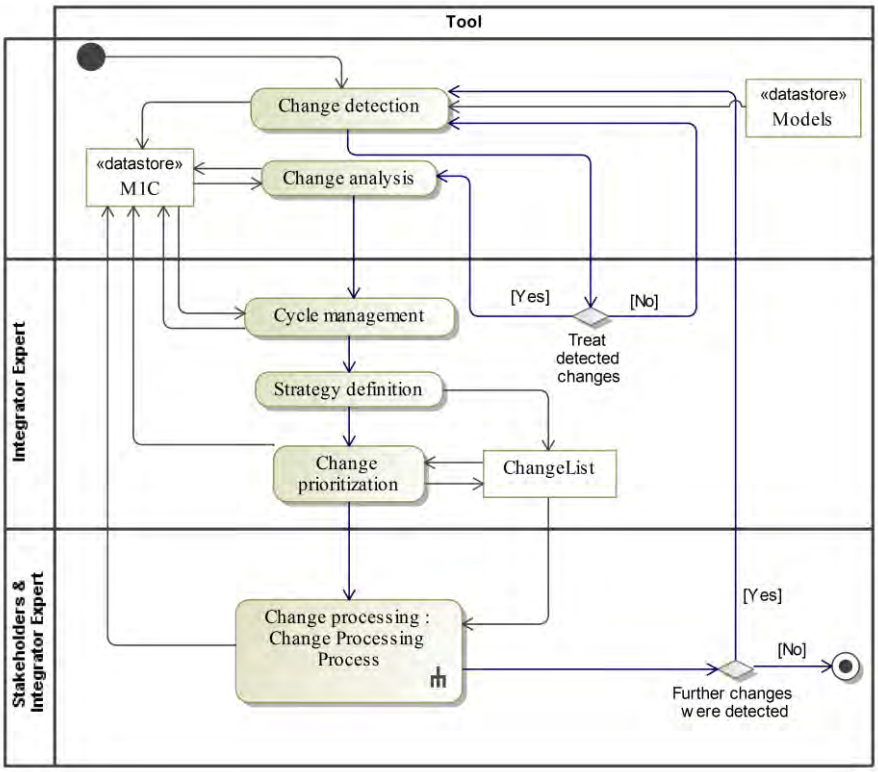


Figure V-2 : Processus de maintien de la cohérence des modèles partiels

V.3. Phase 1 : Détection des changements

La phase de détection des changements a pour objectif d'identifier les changements apportés aux modèles lors de leur évolution, tel que l'ajout d'un élément, sa modification ou sa suppression. Dans la section suivante (V.3.1), nous explicitons la manière de représenter ces changements dans le métamodèle MMC. La façon dont ces changements sont détectés est traitée dans la section V.3.2.

V.3.1. Extension du métamodèle MMC

Afin de représenter ces évolutions, nous avons étendu le MMC (voir le premier encadré de la Figure V-4) en introduisant les concepts suivants :

- *History* : méta-classe utilisée pour garder trace des changements appliqués,
- *DiffElt* : méta-classe abstraite qui permet d'enregistrer, une fois spécialisée, la trace des éléments ayant évolué. Elle possède deux attributs. Le premier contient le type de classification du changement (cf. section V.4) et le second contient la référence de l'élément construite à partir du nom de l'élément, de son méta-élément et du nom du modèle,
- *DeletedElt* : méta-classe qui représente un élément qui n'existe plus dans le modèle initial, mais qui est maintenu pour que sa trace soit conservée,
- *AddedElt* : méta-classe représentant le nouvel élément de modèle ajouté,

- *ModifiedElt* : méta-classe représentant un élément de modèle résultant d'une opération de modification d'un élément du modèle initial.

V.3.2. Enrichissement du M1C

L'extension du métamodèle MMC permet de mémoriser les différents changements sans définir comment détecter ces changements. Ce point fait l'objet de cette section.

Une solution pour résoudre le problème de détection automatique des changements est d'utiliser des techniques de calcul des discordances qui produisent un delta (Δ) entre le modèle initial et le modèle ayant évolué. Comme exemple de Framework intégrant ces techniques, on peut citer *EMFCompare (Brun et Pierantonio, 2008)*, qui fournit un algorithme générique pour le calcul des différences entre deux versions d'un modèle en se basant sur les techniques structurelles de calcul de distances. En se basant sur le delta entre deux modèles, on peut identifier les évolutions et les ajouter au modèle de correspondance (cf. Figure V-3).

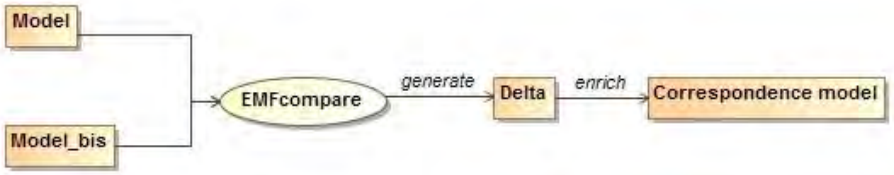


Figure V-3 : Identification des changements avec *EMFCompare*

Cependant, cette approche qu'on peut qualifier d'approche en «temps différé» étant donné que l'information sur les changements se fait à la demande et non en temps réel, possède certains inconvénients qui la rendent lourde à mettre en œuvre :

- Restriction sur le nombre de modèles, étant donné que la comparaison ne peut être faite à la fois que sur un modèle (*Model*) et sa version précédente (*Model_bis*). De ce fait, pour un nombre n de modèles susceptibles d'avoir subi un changement, n deltas sont créés,
- Nécessité de conserver la version initiale de chaque modèle et celle qui a été modifiée,
- Appel de la procédure d'enrichissement du M1C pour chaque delta généré.

Pour remédier à ces inconvénients, nous proposons une solution fondée sur une approche en temps réel de détection automatique des changements. Cette solution consiste à utiliser le patron de conception Observateur (*Observer*)(Gamma et al., 1994, Larman, 2005). Le deuxième encadré de la Figure V-4 présente l'application de ce patron en l'intégrant au MMC. Le concept *ModelElement* représente le sujet concret à observer. Ce dernier est une spécialisation du concept *Subject* qui possède trois méthodes. Deux d'entre elles (*attach* et *detach*) permettent de fixer ou de détacher un objet observateur d'un élément de modèle. La troisième méthode (*notify*) permet de notifier le M1C des changements qui ont eu lieu. La méthode *update* de la méta-classe *Observer* est utilisée lors de la phase de traitement des changements afin de maintenir la cohérence des modèles. L'implémentation de ce patron de conception permet, en temps réel, d'informer le modèle de correspondance des différentes manipulations apportées aux modèles du domaine d'application.

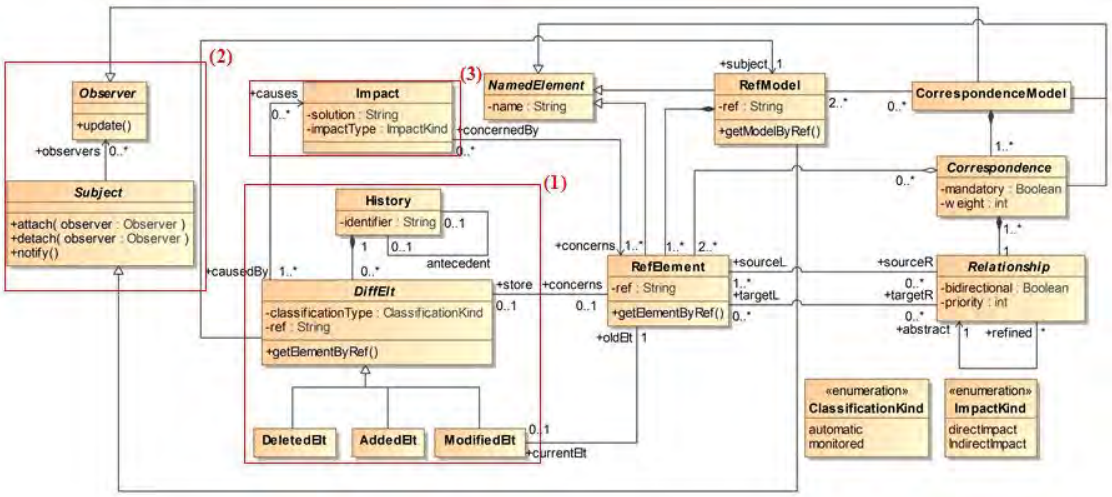


Figure V-4 : Extension du métamodèle de correspondance MMC

Pour des raisons de clarté de présentation et afin de ne pas surcharger le M1C présenté sur la Figure IV-21 du chapitre précédent, nous avons décidé de faire abstraction de la syntaxe concrète et de représenter le M1C par le Tableau V-1 afin de suivre plus facilement les différentes phases du processus de maintien de la cohérence.

Le tableau ci-dessous se lit de la façon suivante. La première ligne par exemple signifie que la correspondance C1, non obligatoire (*mandatory* = *false*), relie, par le type de relation *Similarity* avec une valeur de priorité de 1, l'élément *author* (de type *StereotypedEntity*) du modèle de conception logicielle avec l'élément *AuthorTable* (de type *Table*) du modèle de persistance.

Nous avons attribué, lors de la mise en correspondance des méta-éléments (Section IV.6), les valeurs de priorité 1, 1, 2, 2, 3, 4, 5 respectivement aux types de relation *Similarity*, *Dependency*, *Aggregation*, *Generalization*, *Play*, *Contribution* et *Requirement*.

[1] *Software Design* [2] *Business Process* [3] *Persistence*

Correspondance		**Type de relation**		**Elément de modèle**		
Nom	Obligatoire	Nom	Priorité	SD[1]	BP[2]	PS[3]
C1	Non	*Similarity*	1	*StereotypedEntity:Author*		*Table:AuthorTable*
C2	Non	*Similarity*	1	*Entity:Paper*		*Table:ArticleTable*
C3	Oui	*Equality*	1	*Property:organization*		*Column:organization*
C4	Oui	*Equality*	1	*Property:address*		*Column:address*
C5	Non	*Similarity*	1	*Entity:Reviewer*		*Table:ReviewerTable*
C6	Non	*Aggregation*	1	*Property:firstName,* *Property:lastName*		*Column:fullName*

C7	Non	*Aggregation*	1	*Property:details*		*Column:reviews, Column:decision*
C8	Oui	*Equality*	1	*Property:submissionDate*		*Column:submissionDate*
C9	Non	*Similarity*	1	*Entity:Review*	*DataObject:reviews*	
C10	Oui	*Requirement*	5		*Task:logIn*	*Column:password, Column:e-mail*
C11	Oui	*Contribution*	4	*Operation:reviewPaper()*	*Task:Edit review*	
C12	Oui	*Contribution*	4	*Operation:reviewPaper()*	*Task:Outsource paper*	
C13	Oui	*Contribution*	4	*Operation:createCom()*	*Task:Enter comment*	
C14	Non	*Similarity*	1	*Entity:Comment*	*DataObject:comment*	
C15	Non	*Play*	3	*StreotypedEntity:Reviewer*	*Pool:Reviewer*	
C16	Non	*Similarity*	1		*DataObject:review*	*Column:reviews*

Tableau V-1 : Représentation simplifiée sous forme de tableau du M1C

La première partie du Tableau V-2 illustre certaines évolutions effectuées sur les modèles du CMS. Ces évolutions sont automatiquement détectées via l'outil et ajoutées au M1C, en fonction du type du changement. Premièrement, nous supposons que l'architecte de procédé a ajouté les rôles de président (*Chairman*) et d'auteur (*Author*) au modèle du processus métier. Deuxièmement, nous faisons l'hypothèse que l'architecte logiciel et l'architecte de procédé ont supprimé des éléments de modèle. Le premier a supprimé l'élément *Property:phoneNumber* ainsi que l'élément *Entity:Comment* alors que le deuxième a supprimé l'élément *Task:outsource paper*. Troisièmement, nous supposons que l'architecte logiciel a changé l'élément *createCom( )* de type *Operation* par l'élément *createComments( )* et qu'il a aussi remplacé l'élément *Review* de type *Entity* par l'élément *Note*.

Le mode de traitement d'un changement est fortement lié à son type et aux éléments affectés. La deuxième phase du processus, détaillée dans la section suivante, consiste à analyser les changements, en complétant le M1C (deuxième partie du Tableau V-2).

V.4. Phase 2 : Analyse des changements

Une fois que le M1C a été enrichi par les changements qui ont eu lieu, le rôle de cette phase est de compléter les informations manquantes dans le M1C, à savoir le type de changement et les éléments susceptibles d'être affectés afin d'adapter le traitement des changements.

Tout d'abord une classification des changements en modes est réalisée par l'outil. Ces modes de gestion des modifications visent à gérer au mieux les conséquences d'un changement en attribuant à chaque mode des actions spécifiques. On distingue deux modes : automatique et guidé.

- Mode automatique (*Automatic Evolution*) : il concerne les changements qui mènent à des actions automatiques effectuées sur les modèles. Les types d'actions concernés sont l'ajout et la suppression,
- Mode guidé (*Monitored Evolution*) : le type d'action concerné par ce mode est la modification. Lorsqu'un élément est modifié, la correspondance doit être réévaluée en fonction de la sémantique du type de relation utilisée. Selon (Cicchetti et Ciccozzi, 2013), lorsque la sémantique du type de relation entre en jeu, les problèmes de gestion de versions deviennent plus complexes et ne peuvent pas être traitées automatiquement. En d'autres termes, le support d'automatisation est réduit et les changements sont susceptibles de faire intervenir une assistance humaine afin de décider de la répercussion du changement.

La seconde information à définir concerne les éléments du M1C susceptibles d'être affectés par les changements. Ceci est possible grâce à l'extension du MMC, qui permet de trouver pour un élément modifié, la (les) correspondance(s) à laquelle (auxquelles) il appartient et ainsi de retrouver le(s) élément(s) susceptible(s) d'être affecté(s). Pour chaque répercussion, on spécifie si elle est de type direct ou indirect. Le premier type concerne les éléments directement reliés à l'élément modifié par une correspondance. Le second type est celui des éléments affectés en cascade. Supposons que l'on ait deux correspondances (Figure V-5), la première reliant un élément A à un élément B par la relation Rx et la deuxième un élément B à un élément C par le la relation Ry. Si l'élément A est modifié, l'élément B peut être influencé directement par ce changement, ce qui peut entrainer une répercussion en «cascade» (indirecte) sur C.

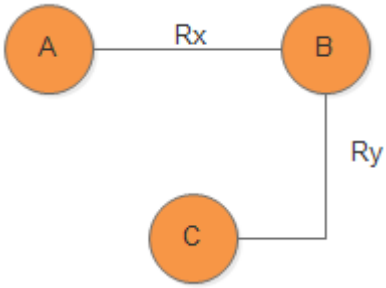


Figure V-5 : Exemple d'influence en cascade

L'information sur le mode de classification apparaît sur le MMC (enrichi) dans l'attribut *classificationType* du concept *DiffElt e*t le type de répercussion dans l'attribut *impactType* du concept *Impact*.

Dans le Tableau V-2, la partie 2 montre les informations ajoutées automatiquement au M1C durant cette phase. Les éléments de modèles ajoutés sont classés en mode automatique et le fait de les ajouter n'a aucune incidence sur la cohérence du M1C. Les éléments supprimés sont eux aussi classés en mode automatique. Par exemple, la suppression de l'élément *Property:phoneNumber* n'a aucune répercussion sur le M1C étant donné qu'il ne participe à aucune des correspondances. Par contre, la suppression de l'élément *Entity:Comment* a une influence directe sur l'élément *DataObject:comment* tandis que la suppression de l'élément *Task:Outsource paper*

a une influence directe sur l'élément *Operation:reviewPaper()* et une influence indirecte sur l'élément *Task:Edit review*.

Les éléments modifiés sont classés en mode guidé. Considérons les modifications représentées dans les deux dernières lignes du tableau. La première possède une influence directe sur l'élément *Task:Enter comment* alors que la seconde modification possède une influence directe sur les éléments *DataObject:review* et *Column:reviews* et une influence indirecte sur les éléments *DataObject:review* et *Property:details*.

[1] Type de Changement [2] Ajout [3] Modification [4] Suppression [5] Mode de Classification [6] Automatique

Partie 1			**Partie 2**		
Type de Chgt.[1]	**Elément de modèle**		**Mode de Class.[5]**	**Elément(s) influencé(s)**	
				Influence directe	**Influence Indirecte**
A.[2]	*Pool:Chairman*		Auto.[6]	Aucune	Aucune
A.	*Pool:Author*		Auto.	Aucune	Aucune
S.[3]	*Task:Outsource paper*		Auto.	*Operation:reviewPaper()*	*Task:Edit review*
S.	*Property:phoneNumber*		Auto.	Aucune	Aucune
S.	*Entity:Comment*		Auto.	*DataObject:comment*	Aucune
M.[4]	Ancien	*Operation:createCom()*	Guidé	*Task:Enter comment*	Aucune
	Nouveau	*Operation:createComments()*			
M.	Ancien	*Entity:Review*	Guidé	*DataObject:review,* *Column:reviews*	*DataObject:review,* *Property:details*
	Nouveau	*Entity:Note*			

Tableau V-2 : Exemple de changements et de leurs influences potentielles

V.5. Phase 3 : Gestion des cycles

Une fois qu'on a connaissance des changements et des influences directes et indirectes, une deuxième analyse est réalisée. Elle consiste à détecter automatiquement les cycles existant après l'établissement de correspondances en examinant les impacts. C'est ce que nous considérons comme étant une «influence en cascade cyclique». En reprenant l'exemple précédent, supposons que l'on ait trois correspondances (Figure V-6). La première relie un élément A à un élément B, la deuxième relie un élément B à un élément C et la troisième relie un élément C à l'élément A. Si

l'élément A est modifié, l'élément B peut être influencé directement par ce changement ce qui influence indirectement l'élément C. C'est dans le cas où l'élément C est relié à l'élément A qui est le responsable de l'impact qu'on parle d'influence en cascade cyclique. Les différentes influences en cascade cycliques sont remontées à l'expert intégrateur. Il devra décider de la suppression d'une de ces correspondances afin de rompre le cycle.

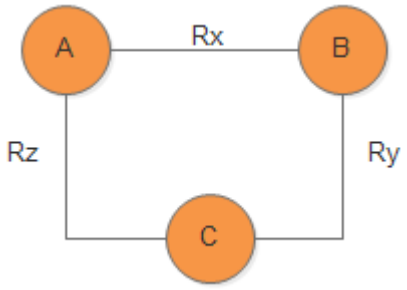


Figure V-6 : Exemple d'influence en cascade cyclique

V.6. Phase 4 : Définition de la stratégie d'ordonnancement des changements

Le but de cette étape est de produire une liste ordonnée de changements selon la stratégie à choisir. Nous proposons deux stratégies : une stratégie fondée sur le mode de classification (*Classification Based Strategy*) et une stratégie fondée sur le type d'influence (*Impact Based Strategy*). La stratégie fondée sur le mode de classification consiste à créer une liste des changements qui contient dans l'ordre, les changements classifiés en mode automatique suivis de ceux en mode guidé (ou l'inverse, selon le choix de l'expert en fonction des changements détectés). Par exemple, en choisissant la stratégie à base de mode de classification automatique, la liste de changements produite est la suivante : *Pool:Chairman, Pool:Author, Task:Outsource paper, Property:phoneNumber, Entity:Comment; Operation:createComments(), Entity:Note*. La seconde stratégie se base sur la production d'une liste de changements qui commence par les changements qui ont une influence directe et indirecte suivi de ceux qui ont une influence directe uniquement (ou l'inverse, selon le choix de l'expert en fonction des changements détectés). Par exemple, en choisissant la stratégie à base d'influence directe et indirecte, la liste de changements produite est la suivante : *Entity:Note, Task:Outsource paper; Pool:Chairman, Pool:Author, Property:phoneNumber, Entity:Comment, Operation:createComments()*.

Supposons que l'expert ait choisi une stratégie à base de type de classification en mode automatique conformément au Tableau V-2 présenté dans la section précédente. On a ici plusieurs éléments en mode automatique (et plusieurs autres en mode guidé). La question qui se pose est lequel d'entre eux doit être traité en premier ? La réponse à cette question est abordée dans la section suivante.

V.7. Phase 5 : Priorisation des changements

L'ordre de traitement des évolutions de modèles est important car l'impact sur le système dépend de cet ordre dans le sens où le résultat diffère selon l'ordre choisi. Nous avons mis en place des coefficients de pondération (attribut *weight*) associés aux correspondances pour

permettre de déterminer un ordre de traitement des changements. En prenant l'exemple de la Figure V-6, en supposant que les éléments A et C aient été modifiés, la question à se poser est quel changement traiter en premier sachant que les deux ont une influence sur le même élément B ? et faudra-t-il traiter les deux changements ? La priorisation permet de répondre à cette question en traitant d'abord le changement qui a le coefficient de pondération le plus élevé. Le changement qui suit doit tenir compte de l'influence traitée par le changement précédent. Autrement dit, si le changement de l'élément A nécessite la modification de l'élément B, l'élément C ne peut modifier l'élément B que si la correspondance reliant l'élément A et B n'est pas remise en cause.

La définition de l'ordre du traitement se fait en calculant le coefficient de pondération de la correspondance. Ce coefficient se calcule par la formule suivante :

$$\text{weight} = \sum_{k=0}^{n} (ElementAffectéDirectement_{k} * priority) + \sum_{k=0}^{n} (ElementAffectéInDirectement_{k} * priority)$$

Le Tableau V-3 illustre le résultat de la phase de priorisation des changements. Il prend la forme d'une nouvelle liste de changements ordonnée en fonction de la stratégie choisie dans l'étape précédente (classification en mode automatique) et du coefficient calculé. Après calcul des coefficients de pondération de chaque correspondance, les changements sont traités dans l'ordre suivant :

1. *Task:Outsource paper (weight=8)*,
2. *Entity:Comment (weight=1)*,
3. *Pool:Author (weight=0),*
4. *Pool:Chairman (weight=0)*,
5. *Entity:Note* (weight=5),
6. *Operation:createComments()* (weight=4).

Les deux derniers éléments sont traités à la fin même s'ils possèdent un coefficient élevé pour la simple raison que la stratégie choisie traite le mode automatique avant le mode guidé.

Lorsque plusieurs changements ont le même coefficient de pondération, c'est à l'expert de trancher sur l'ordre de traitement, sauf dans le cas des changements de type ajout (comme c'est le cas des deux premiers éléments du tableau ci-dessous), où la préférence est quelconque étant donné que tous les éléments ajoutés sont traités à la fin du processus de traitement des changements par le processus de mise en correspondance (cf. Figure V-7).

[1] Type de Changement [2] Ajout [3] Modification [4] Suppression [5] Mode de Classification [6] Automatique

Type de Chgt.[1]	Elément de modèle		Mode de Class.[5]	Coefficient de pondération	Ordre
A.[2]	*Pool:Chairman*		**Auto.[6]**	0	4
A.	*Pool:Author*		**Auto.**	0	3
S.[3]	*Task:Outsource paper*		**Auto.**	Weight = Operation:reviewPaper() * priorityContribution+Task:Edit review*priorityContribution Weight = 1*4+1*4 = 8	1
S.	*Entity:Comment*		**Auto.**	Weight = DataObject:comment * proiritySimilarity Weight = 1*1 = 1	2
M.[4]	Ancien	*Operation:createCom()*	**Guidé**	Weight = Operation:createComments() * priorityContribution Weight = 1*4 =4	6
	Nouveau	*Operation:createComments()*			
M.	Ancien Nouveau	*Entity:Review* *Entity:Note*	**Guidé**	Weight = DataObject:review * prioritySimilarity + Column:reviews * prioritySimilarity + DataObject:review * prioritySimilarity + Property:details * priorityAggregation + Column:reviews*proiritySimilarity Weight = 1*1+1*1+1*1+1*2+1*1=6	5

Tableau V-3 : Ordonnancement des changements et calcul de la priorité

V.8. Phase 6 : Traitement des changements

C'est dans cette dernière phase que les modèles source et le M1C risquent d'évoluer pour tenir compte des changements qui ont été détectés, par application d'un ensemble d'actions (directives). La Figure V-7 présente le processus mis en oeuvre.

Les changements classés en mode automatique sont traités automatiquement. Dans le cas où un élément a été ajouté, le processus de mise en correspondance est relancé à la fin du processus de traitement des changements pour pouvoir tenir compte des ajouts effectués et ne pas relancer la mise en correspondance pour chaque nouvel élément. Dans le cas où un élément a

été supprimé, toute correspondance impliquant cet élément devient orpheline. Nous définissons une correspondance orpheline comme une correspondance pour laquelle une de ses extrémités − un élément de modèle - est manquante. Quand une correspondance est orpheline, on vérifie si elle est obligatoire pour le système concerné (valeur de l'attribut *mandatory*). Si c'est le cas, l'élément supprimé est restauré, sinon la correspondance est supprimée du modèle de correspondance.

Les autres changements, classés en mode guidé, sont gérés de façon semi-automatique. La correspondance est maintenue si, après le changement d'une de ses extrémités, elle reste toujours valide vis-à-vis de la sémantique associée au type de la relation. Cette validité est vérifiée en exécutant le corps de la condition associé au type de la relation (voir Section IV.7.3.2 ci-avant). Dans le cas contraire, quand un élément est modifié, il faut a priori modifier chacun des éléments auxquels il est relié, sous réserve bien sûr que leur modification soit autorisée par les responsables concepteurs des modèles en question. Si la modification de l'élément de modèle n'est pas autorisée, la correspondance est supprimée dans le cas où elle n'est pas obligatoire (*mandatory* = *false*). Par contre si elle est essentielle pour le système, une décision doit être prise par l'expert intégrateur en collaboration avec les acteurs des modèles incluant les éléments concernés, afin de décider des solutions possibles. Parmi ces solutions on peut citer : la modification de l'élément, la modification de l'élément de l'extrémité de la correspondance.

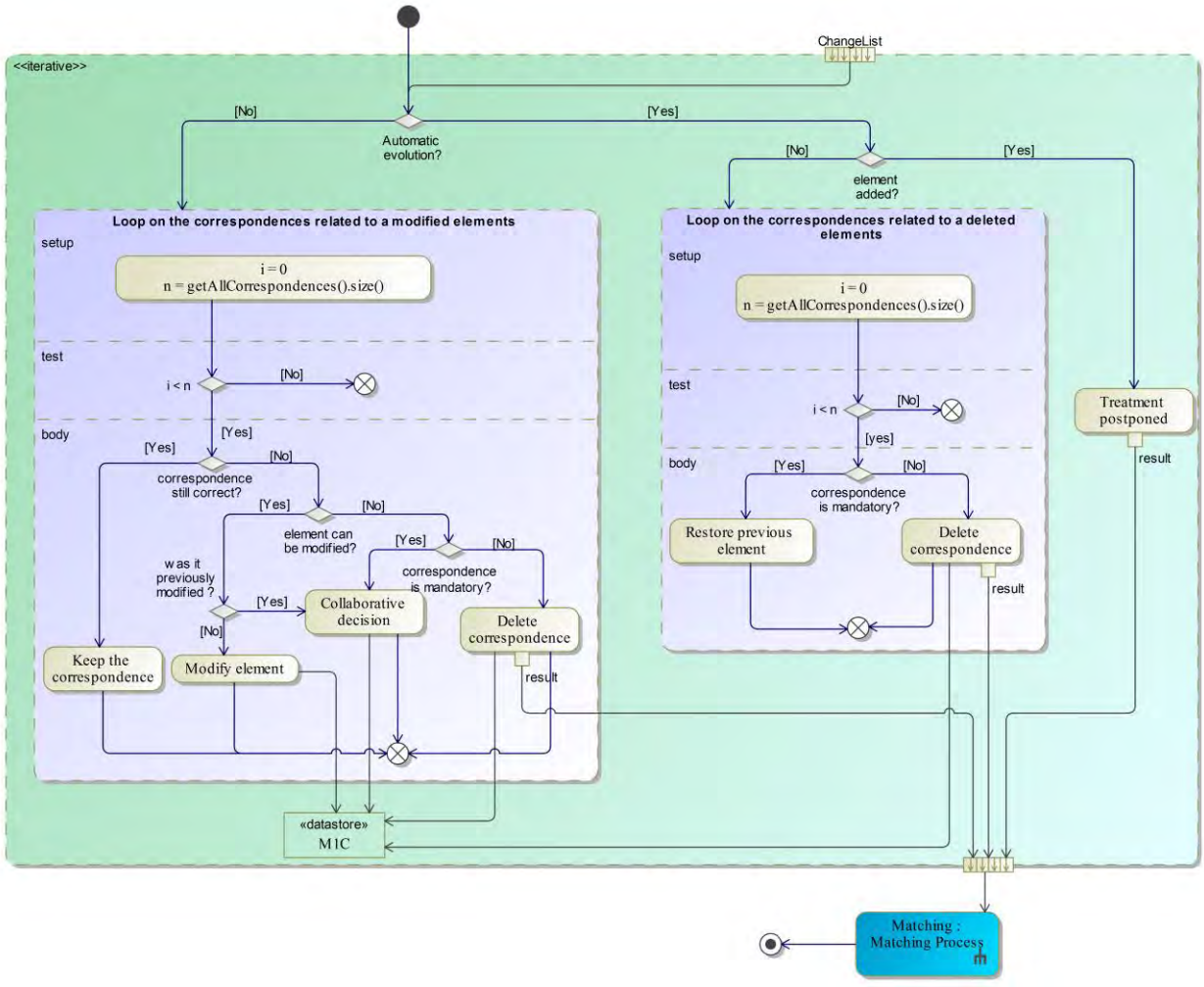


Figure V-7 : Processus de traitement des changements

Concernant notre exemple fil conducteur, comme cela est décrit dans le Tableau V-4 résultant de l'application du processus de la Figure V-7 précédente sur la liste des changements (représenté dans le Tableau V-3), le premier élément supprimé est restauré étant donné que la correspondance est obligatoire (*mandatory* = *true*). La correspondance impliquant le deuxième élément est supprimée, vu qu'il s'agit d'une correspondance orpheline. L'opération de mise en correspondance est invoquée à la fin du processus pour rechercher d'éventuelles correspondances pour le troisième et le quatrième élément. Deux correspondances utilisant le type de relation *play* sont ainsi créées (cf. Tableau V-5). Le cinquième élément de la liste est impliqué dans deux correspondances (cf. Tableau V-2). La première le relie à l'élément *DataObject:review* et la seconde à l'élément *Column:reviews*. L'élément de l'extrémité opposée de la première correspondance est modifié étant donné que la correspondance n'est plus correcte. Concernant la seconde correspondance, étant donné que l'élément opposé ne peut pas être modifié (en raison du choix d'interdire la modification ou la suppression des éléments du modèle de persistance) et que la correspondance n'est pas obligatoire, elle est supprimée.

[1] Type de Changement [2] Ajout [3] Modification [4] Suppression [5] Type de Classification [6] Automatique

Ordre	**TChgt.[1]**	**Elément de modèle**		**TClass.[5]**	**Action effectuée**
1	S.[3]	Task:Outsource paper		Auto.	Restauration de l'élément supprimé
2	S.	Entity:Comment		Auto.	Suppression de la correspondance
3	A.[2]	Pool:Author		Auto.	Mise en correspondance
4	A.	Pool:Chairman		Auto.[6]	Mise en correspondance
5	M.	Ancien	*Entity:Review*	Guidé	Pour la première correspondance, modification de l'élément de l'extrémité. Pour la seconde correspondance sa suppression.
		Nouveau	*Entity:Note*		
6	M.	Ancien	*Entity:createCom()*	Guidé	Maintien de la correspondance
		Nouveau	*Entity:createComments*		

Tableau V-4 : Actions appliquées pour chaque évolution de modèle du CMS

Le Tableau V-5 présente le nouvel état du M1C après application des actions relatives aux différents changements. Les lignes encadrées montrent les changements qui ont été faits par rapport au M1C présenté dans le Tableau V-1. Les lignes en soulignement double (correspondances : C17 et C18) représentent les éléments ajoutés, celles en soulignement simple (correspondances : C14 et C16) les éléments modifiés. Les éléments supprimés sont barrés.

[1] *Software Design* [2] *Business Process* [3] *Persistence*

Correspondance		**Type de relation**		**Elément de modèle**		
Nom	Obligatoire	Nom	Priorité	SD[1]	BP[2]	PS[3]
C1	Non	*Similarity*	1	*StereotypedEntity:Author*		*Table:AuthorTable*
C2	Non	*Similarity*	1	*Entity:Paper*		*Table:ArticleTable*
C3	Oui	*Equality*	1	*Property:organization*		*Column:organization*
C4	Oui	*Equality*	1	*Property:address*		*Column:address*
C5	Non	*Similarity*	1	*Entity:Reviewer*		*Table:ReviewerTable*
C6	Non	*Aggregation*	2	*Property:firstName, Property:lastName*		*Column:fullName*
C7	Non	*Aggregation*	2	*Property:details*		*Column:reviews, Column:decision*
C8	Oui	*Equality*	1	*Property:submissionDate*		*Column:submissionDate*
C9	Non	*Similarity*	1	*Entity:Note*	*DataObject:notice*	
C10	Oui	*Requirement*	5		*Task:logIn*	*Column:password, Column:e-mail*
C11	Oui	*Contribution*	4	*Operation:reviewPaper()*	*Task:Edit review*	
C12	Oui	*Contribution*	4	*Operation:reviewPaper()*	*Task:Outsource paper*	
C13	Oui	*Contribution*	4	*Operation:createComents()*	*Task:Enter comment*	
~~C14~~	Non	~~*Similarity*~~	~~1~~	~~*Entity:Comment*~~	~~*DataObject:comment*~~	
C15	Non	*Play*	3	*StreotypedEntity:Reviewer*	*Pool:Reviewer*	
~~C16~~	Non	~~*Similarity*~~	~~1~~		~~*DataObject:review*~~	~~*Column:reviews*~~
C17	Non	*Play*	3	*StreotypedEntity:Author*	*Pool:Author*	
C18	Non	*Play*	3	*StreotypedEntity:Chairman*	*Pool:Chairman*	

Tableau V-5 : Aperçu de la nouvelle version du M1C

V.9. Conclusion

Durant le cycle de modélisation par points de vue d'un système complexe, la description des modèles évolue fréquemment pour s'adapter aux besoins métier et à l'apparition de nouvelles exigences et contraintes. Dans un environnement de multi-modélisation, plusieurs changements peuvent être effectués en même temps sur différents modèles. Pour gérer la cohérence entre ces modèles, nous avons proposé un métamodèle de correspondance adapté à la description des évolutions et un processus de gestion d'évolution permettant d'identifier les changements, d'identifier les répercussions sur les éléments de modèles concernés, de gérer les cycles, de traiter les changements et ainsi de garantir le maintien de la cohérence du modèle global. Le chapitre suivant est consacré au prototype développé.

CHAPITRE VI. PROTOTYPE HMCS

"Technology is the effort to save effort"

— *José Ortega y Gasset*

VI.1. Introduction

Ce chapitre traite de l'outil développé comme support à l'approche proposée dans cette thèse. L'objectif est de prouver la faisabilité des concepts et de valider la démarche introduite dans les chapitres précédents en utilisant l'exemple didactique du CMS. Dans cette optique, nous avons développé un Outil Support appelé HMCS (*Heterogeneous Matching and Consistency management Suite*) sur la plate-fome Eclipse. Il s'agit d'un ensemble de plug-ins fournissant aux utilisateurs deux fonctionnalités majeures. La première concerne l'établissement des correspondances entre les modèles hétérogènes, tandis que la seconde a pour objectif de maintenir leur cohérence lorsqu'ils évoluent.

La suite de ce chapitre est organisée comme suit : la section VI.2 présente la plate-forme Eclipse et les outils de manipulation de modèles. L'architecture technique et les Frameworks utilisés sont présentés dans la section VI.3. Nous présentons dans la section VI.4 l'enchainement fonctionnel et les détails de l'implémentation du prototype, et nous concluons par la section VI.5 en faisant un bilan du prototype et de ses limites.

VI.2. La plateforme Eclipse

VI.2.1. Vue d'ensemble d'Eclipse

Eclipse est une communauté open source dont les projets sont axés sur la construction d'une plate-forme de développement. La plateforme Eclipse, initiée par IBM, est un atelier de génie logiciel (AGL) conçu dans le but de fournir une plateforme modulaire et extensible pour la construction, le déploiement et la gestion des logiciels, couvrant l'ensemble de leurs cycles de vie. Tout le code de la plateforme a été mis à la disposition de la communauté open source afin de contribuer à son développement.

Le graphique suivant (Figure VI-1), conçu par Holger Voorman, indique le nombre de projets et de lignes de code sous les différentes distributions d'Eclipse au fil des ans. Avant 2012, la plateforme Eclipse a été publiée dans la version 3.x sous plusieurs distributions parmi lesquelles Helios (3.6), Indigo (3.7), etc. À partir de 2012, la version de la plateforme est passée au 4.x, avec les distributions Juno (4.2), Kepler (4.3) et Luna (4.4) sortie en 2014.

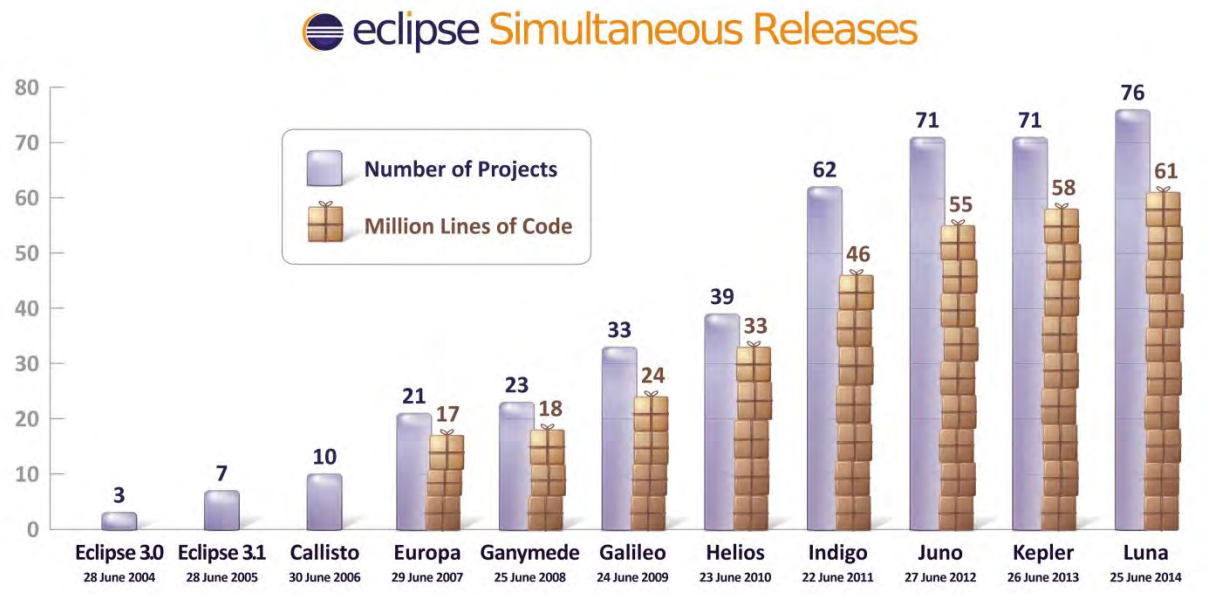


Figure VI-1 : Croissance de la plateforme Eclipse dans le temps (Voormann, 2014)

La Figure VI-2 montre les composants de la version 4.x. Elle introduit l'E4AP (*Eclipse 4 Application Plateform*) qui se base sur de nouvelles technologies à savoir :

- Interface utilisateur à base du modèle EMF (*Eclipse Modeling Framework*) qui sera utilisée par la suite pour générer le code SWT (*Standard Widget Toolkit*),
- Utilisation des feuilles de style CSS (*Cascading Style Sheets*) pour changer facilement l'apparence et la convivialité de la plateforme Eclipse,
- Modèle d'injection de dépendances en utilisant les annotations de la JSR 330[8],
- Modèle de programmation à base de services.

L'E4AP contient aussi quatre composants. Le premier est une implémentation de la spécification OSGi (*Open Service Gateway initiative*) : Equinox. Elle fournit le cadre nécessaire pour exécuter une application Eclipse modulaire. Le deuxième composant est EMF qui est utilisé pour la manipulation des modèles et les deux derniers sont les bibliothèques graphiques SWT et JFace.

Au-dessus de l'EA4P, la version 4.x définit un workbench qui fournit le Framework pour l'application. Il est responsable de l'affichage de tous les autres composants de l'interface utilisateur. Etant donné que la majorité des plug-ins fonctionnent encore sous la version 3.x, une

[8] https://jcp.org/en/jsr/detail?id=330

couche de compatibilité a été définie afin de les exécuter sous la version 4.x, sans modification de leurs contenus (Vogel, 2014).

Finalement, à l'instar de la version 3.x, la version 4.x comprend un ensemble de composants : outil de développement Java (*Java Developement Tooling*), environnement de développement de plug-ins (*Plug-in Development Environment*), support de contrôle de versions, etc.

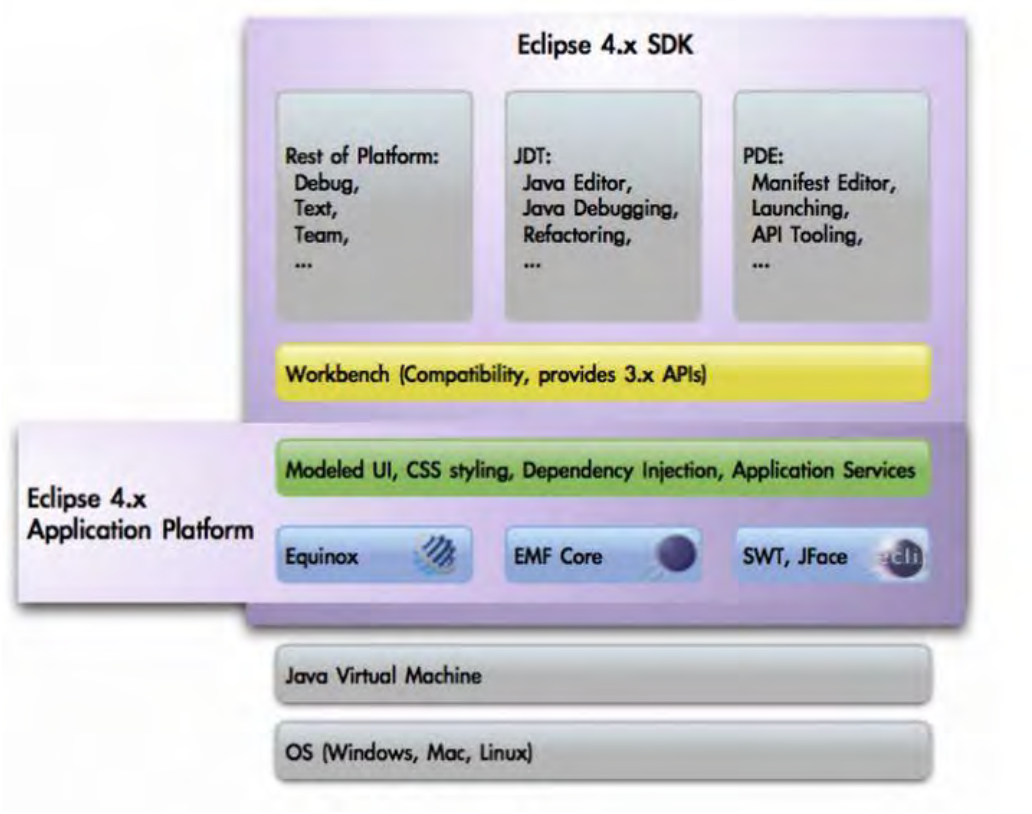


Figure VI-2 : Architecture d'Eclipse 4.x (Eclipse, 2011a)

Après avoir présenté brièvement la plate-forme Eclipse et son architecture, nous allons voir dans les sections qui suivent la manière dont celle-ci a été exploitée pour la manipulation des modèles.

VI.2.2. Le projet Eclipse Modeling Project

Avant de nous plonger dans les détails concernant notre proposition en termes d'architecture technique, nous présentons d'abord un aperçu des projets qui fonctionnent autour d'Eclipse, plus particulièrement ceux de modélisation.

Eclipse propose un ensemble de projets adaptés en fonction du métier des utilisateurs. On peut citer : BIRT, RT, SOA Platform Project, Eclipse Web Tools Platform Projects, Data Tools Platform, Eclipse Modeling Project, etc. La communauté du MDE, s'intéresse plus particulièrement au projet Eclipse Modeling Project (EMP).

Nous proposons sur la Figure VI-3 une présentation de la structure de l'EMP sous forme moléculaire, et sous forme atomique, celle de ses domaines fonctionnels organisés en différents

projets à savoir : Abstract Syntax Development (ASD), Model-to-Model (M2M), Model-to-Text (M2T), Textual Concrete Syntax (TCS), Graphical Concrete Syntax (GCS), Model Development Tools (MDT) et le Eclipse Model Framework Technology (EMFT).

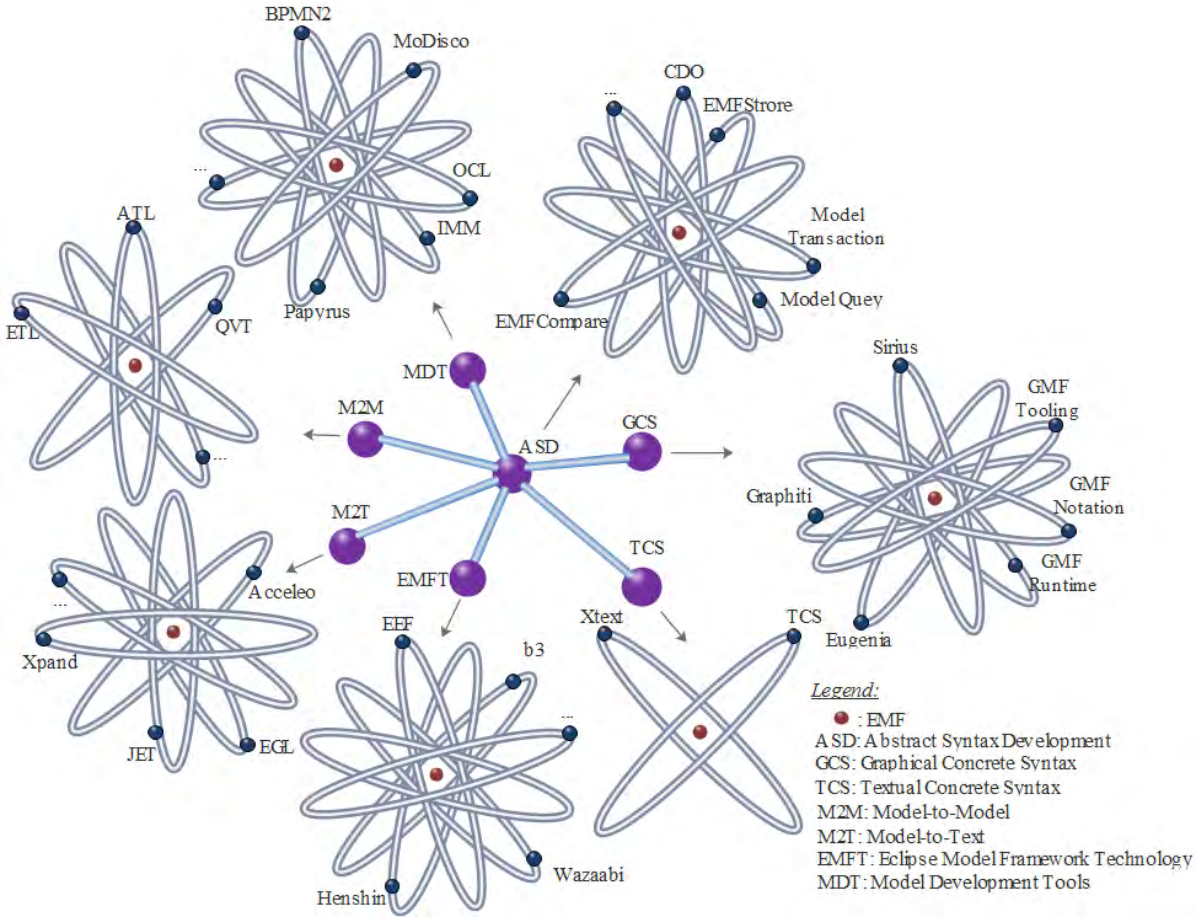


Figure VI-3 : Structuration d'Eclipse Modeling Project

L'élément central de l'EMP est l'ASD. Il fournit les Frameworks nécessaires à la description de la syntaxe abstraite d'un langage, sa structure sémantique et un support pour les transactions, les requêtes, la comparaison, la validation et la persistance. L'ASD est entouré par plusieurs composants regroupés en fonction de leurs objectifs. Tout d'abord, on a les Frameworks de transformation de modèles. L'EMP supporte deux types de transformation. La transformation M2M (Model-to-Model) et la transformation M2T (Model-to-Text). Dans le premier cas, la transformation prend en entrée un modèle et produit un autre modèle en sortie, tandis que dans le second cas, le produit prend la forme d'un texte. Deuxièmement, on a les Frameworks de développement de syntaxe concrète qui sont eux aussi répartis en deux types : TCS et GCS. Les Frameworks de type TCS permettent l'encodage de l'information en utilisant des séquences de caractères comme dans la plupart des langages de programmation. Ceux de type GCS encodent l'information en utilisant des arrangements spatiaux d'éléments graphiques (et textuels) (Moody, 2009). Une autre fonctionnalité assurée par l'EMP est le MDT qui permet de fournir les outils nécessaires pour le support des standards industriels comme ceux de l'OMG (UML, OCL, BPMN, etc.) à titre d'exemple. Finalement l'EMP propose, sous la tutelle du projet EMFT, plusieurs sous-projets de recherche en incubation dans un intérêt exploratoire. Ces

derniers peuvent être, une fois approuvés, ajoutés sous un des projets de modélisation. Ce fut le cas pour Xtext avant son intégration dans le projet TCS.

VI.3. Architecture technique du HMCS

L'architecture (cf. Figure VI-4), basée sur la plate-forme Eclipse, que nous avons adoptée pour le développement du protoype HMCS repose sur un ensemble développés (AMT, RT, FT, etc…), en s'appuyant sur les Frameworks suivants : EMF, KOMMA, GMF, Xtext, JET, EMFCollab, TwoUse et CDO. Dans ce qui suit, nous présentons ces derniers. Le détail sur les modules développés fait l'objet de la section VI.4.

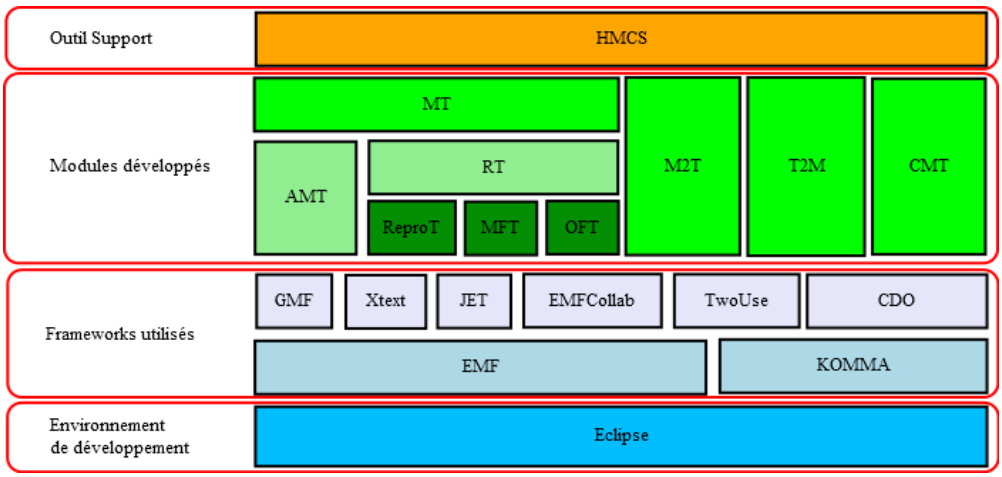


Figure VI-4 : Architecture technique du HMCS

VI.3.1. EMF (Steinberg et al., 2009)

À la base de nombreux outils MDE, EMF est un environnement de modélisation et de génération de code qui facilite la construction d'outils et d'applications basées sur des modèles de données structurées. EMF dispose de son propre langage de méta-modélisation, Ecore, qui est dérivé de la spécification EMOF de l'OMG.

EMF est fondé sur trois briques essentielles[9] :

- EMF.Core : Définit Ecore pour la description des métamodèles. La persistance des métamodèles est effectuée en XMI pour garantir l'interopérabilité avec les autres Frameworks,
- EMF.Edit : Définit un canevas de classes pour le développement d'éditeurs arborescents de modèles EMF,
- EMF.Codegen : Permet de générer le code java nécessaire à la création d'éditeurs pour un modèle EMF.

Pour son fonctionnement, EMF fait appel à deux autres Frameworks d'Eclipse : JET et JMerge. Le premier est utilisé pour générer des fichiers tels que les classes Java alors que le second permet d'éviter d'écraser les modifications apportées par l'utilisateur lors de la

[9] À noter qu'on ne traite pas le mode dynamique (ou réflexif) qui limite les capacités d'EMF

régénération du code. De plus, JET est aussi utilisé dans de nombreux projets industriels tels que BluAge[10] Bonita BPM[11], Rational Software Architect[12], etc.

L'une des faiblesses d'EMF, qu'on comblera avec d'autres Frameworks, est le manque de support de la syntaxe concrète. En effet EMF dispose uniquement d'un éditeur arborescent qui s'appuie sur la structure de contenance des éléments du métamodèle.

VI.3.2. KOMMA (Iwu, 2014)

KOMMA est un Framework fondé sur les technologies du web sémantique dans l'optique de la gestion et de la visualisation des ontologies. Il supporte à la fois le langage OWL (*Web Ontology Language*), RDF (*Resource Description Framework*) et RDFS (*Resource Description Framework Schema*).

Puisque les langages de représentation de connaissances peuvent être considérés comme des langages de modélisation dans le sens du MDE, l'objectif de KOMMA est de combiner les fournisseurs de persistance RDF avec une visualisation souple ressemblant à EMF et avec des fonctionnalités d'édition, au sein d'un Framework d'application unifié pour les systèmes logiciels à base d'ontologies (Wenzel, 2011).

VI.3.3. GMF (Gronback, 2009)

Le Framework GMF (*Graphical Modeling Framework*) est un projet Eclipse pour la génération d'éditeurs graphiques, qui s'appuie sur EMF et GEF (*Graphical Editing Framework*) (Modica et al., 2009).

Afin de pouvoir générer un éditeur graphique, GMF doit avoir en entrée les modèles suivants :

- Modèle de domaine (Domain Model) : Représente la syntaxe abstraite du langage à modéliser. Exprimé en Ecore, ce modèle contient l'ensemble des concepts et des relations qui doivent être implémentés par l'éditeur,
- Modèle de génération (*Domain Gen Model*) : Il permet la génération de l'éditeur arborescent du modèle,
- Modèle de définition graphique (*Graphical Def Model*) : Ce modèle permet la conception de figures (les nœuds, les liens, les compartiments, …) qui seront utilisées pour représenter les éléments du modèle de domaine sur le canevas du futur éditeur,
- Modèle de palette (*Tooling Def Model*) : C'est un modèle d'outillage correspondant au modèle de définition graphique. Il prend en charge la définition d'outils de la palette ainsi que les icônes et labels associés,

[10] http://www.bluage.com/
[11] http://www.bonitasoft.com/
[12] http://www.ibm.com/developerworks/downloads/r/architect/

- Modèle de mapping (*Mapping Model*) : Le modèle de mapping est constitué des liaisons des éléments du modèle de définition graphique, de ceux du modèle d'outillage et des éléments du modèle de domaine correspondants,
- Modèle générateur d'éditeur de diagramme (*Diagram Editor Gen Model*) : Ce modèle est créé suite à une transformation du modèle de «mapping». Une fois obtenu, ce modèle sera utilisé pour la génération du plug-in de l'éditeur en code Java.

VI.3.4. Xtext (Bettini, 2013)

Xtext est un langage de développement d'éditeur textuel s'appuyant sur EMF pour la manipulation de modèles. Il fournit un langage de définition de grammaire similaire à BNF (EBNF : Extended Backus-Naur Form (Wirth, 1996)) pour la description de la syntaxe du langage. L'éditeur Xtext généré propose un ensemble de fonctionnalités : coloration syntaxique, complétion automatique, indentation automatique, navigation, recherche à travers le *outline* et finalement la validation par l'intermédiaire d'un langage d'expression de type OCL nommé Check (Koster, 2007).

VI.3.5. JET (Eclipse, 2011b)

JET est un langage de transformation M2T utilisé non seulement pour les modèles EMF mais aussi pour les objets à base de Java. JET utilise une technologie à base de template, afin de contrôler la sortie générée permettant d'obtenir du texte qui peut prendre la forme d'un code java, c#, xml, html, etc.

La syntaxe de JET est très semblable à la syntaxe des pages JSP[13] (*Java Server Pages*). Comme une JSP est convertie en «servlet» qui sert à créer la réponse qui sera envoyée au navigateur, les templates sont utilisées pour la création d'une classe d'implémentation java par le mécanisme de traduction. Cette dernière est instanciée, avec le mécanisme de génération, afin de produire le résultat désiré sous forme d'une suite de chaines de caractères (texte).

VI.3.6. EMFCollab (Qgears_Kft., 2010)

EMFCollab est un projet open source sous License EPL. Il s'agit d'une solution légère permettant à plusieurs utilisateurs de modifier simultanément un unique modèle EMF. Le serveur contient le modèle qui reste chargé en permanence en mémoire et les clients en stockent aussi une copie dans leurs mémoires. Ce modèle est maintenu en synchronisation sur le réseau ce qui permet à tous les clients de percevoir la même copie principale.

[13] http://java.sun.com/products/jsp/

VI.3.7. TwoUse (Parreiras et al., 2007)

TwoUse est un Framework permettant de réduire l'écart entre les standards W3C du Web sémantique et les standards de l'OMG de l'ingénierie des modèles. Les composants de base de TwoUse permettent :

- L'intégration d'un métamodèle composé à partir des métamodèles UML (incluant OCL) et d'OWL,
- La spécification des requêtes d'exploitation se rapportant au mécanisme de raisonnement d'OWL,
- La définition d'un profil commun pour représenter des modèles hybrides, ainsi que d'autres syntaxes concrètes.

VI.3.8. CDO (Cdo, 2014)

CDO (*Connected Data Objects*) est un Framework de dépôt de modèles. Cet outil de persistance de modèles supporte différentes stratégies pour les bases de données objet, NoSQL et relationnelles. Il est optimisé pour un support transactionnel d'objets volumineux.

VI.4. Enchaînement fonctionnel du HMCS

L'enchaînement fonctionnel du HMCS, représenté par des engrenages sur la Figure VI-5, est composé de deux modules de base (MT: *Matching Tool*, CMT: *Consistency Management Tool*) et de deux modules de support pour les transformations M2T et T2M.

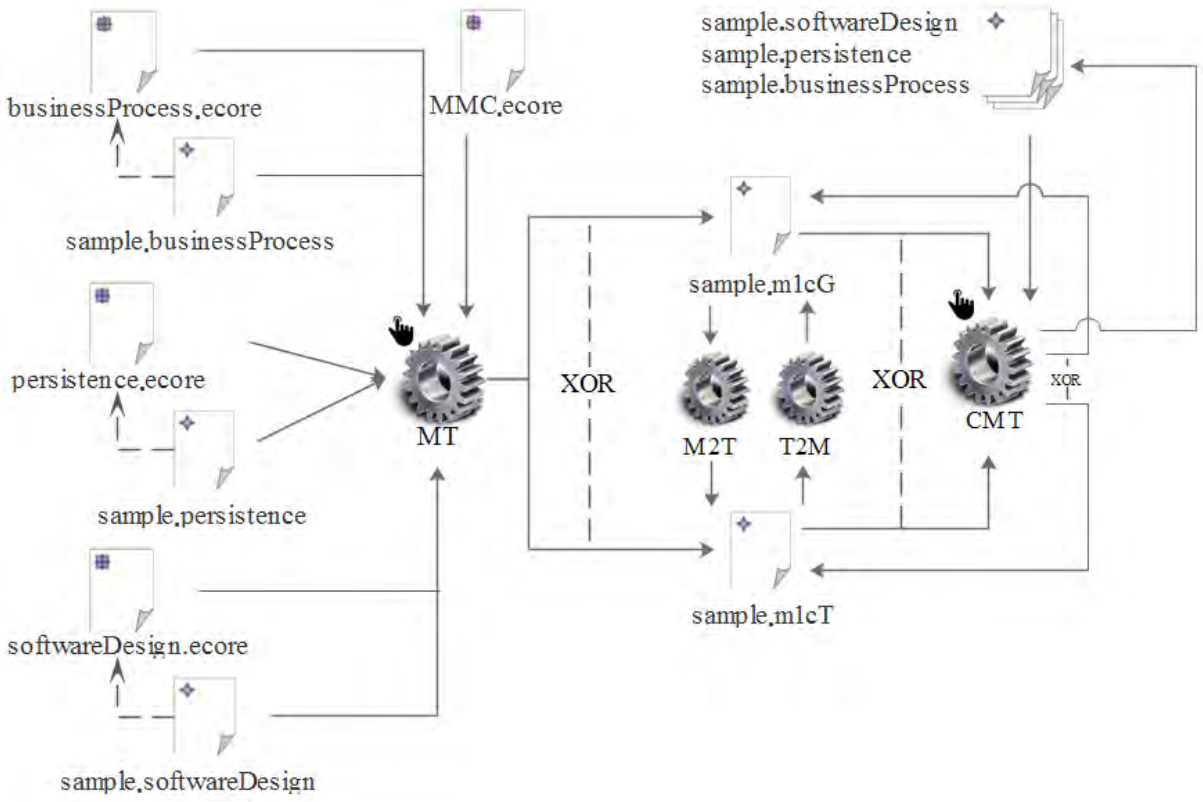


Figure VI-5 : Enchaînement fonctionnel du HMCS

Dans le cas du système CMS présenté dans la section IV.2, le module MT prend en entrée les modèles : sample.businessProcess, sample.persistence et sample.softwareDesign ainsi que les métamodèles auxquels ils sont respectivement conformes : businessProcess.ecore, persistence.ecore et softwareDesign.ecore. Pour mettre en évidence les correspondances entre les différents éléments de modèles de manière adaptée aux différents modes de travail, le module MT produit comme résultat un modèle de correspondance sous forme textuelle (sample.m1cT) ou sous forme graphique (sample.m1cG), selon le besoin de l'expert. Le fait de posséder deux représentations (textuelle et graphique) du modèle de correspondance ne remet pas en cause la cohérence des correspondances. En fait les deux formes de représentation sont synchronisées, c'est-à-dire qu'un acteur peut créer une représentation graphique du modèle de correspondance et continuer sur le même modèle de correspondance mais cette fois-ci en représentation textuelle et inversement.

Ceci est possible grâce aux deux modules de transformations M2T et T2M. Le premier module a pour objectif de «sérialiser[14]», à travers la technologie JET, le modèle sample.m1cG afin de produire la représentation textuelle (sample.m1cT). Le second module a pour objectif de retrouver le modèle graphique en «parsant» la représentation textuelle. La mise en œuvre du module T2M a été codée en Java en s'appuyant sur les bibliothèques de gestion de modèles.

Lorsqu'un ou plusieurs modèles source ont subi un ou plusieurs changements, le modèle de correspondance risque de ne pas être en cohérence avec la réalité. La gestion de la cohérence des modèles source vis-à-vis du modèle de correspondance est supportée par le module CMT.

[14] Opération consistant à transformer un modèle sous forme graphique en un autre sous forme textuelle

VI.4.1. Module Matching Tool (MT)

Matching Tool a été développé sous la forme d'un plug-in Eclipse. Comme représenté par la Figure VI-6, mis à part les modules de transformation (dont le rôle a été expliqué dans la section précédente), il regroupe deux autres modules : AMT (*Assisted Matching Tool*) et RT (*Refining Tool*).

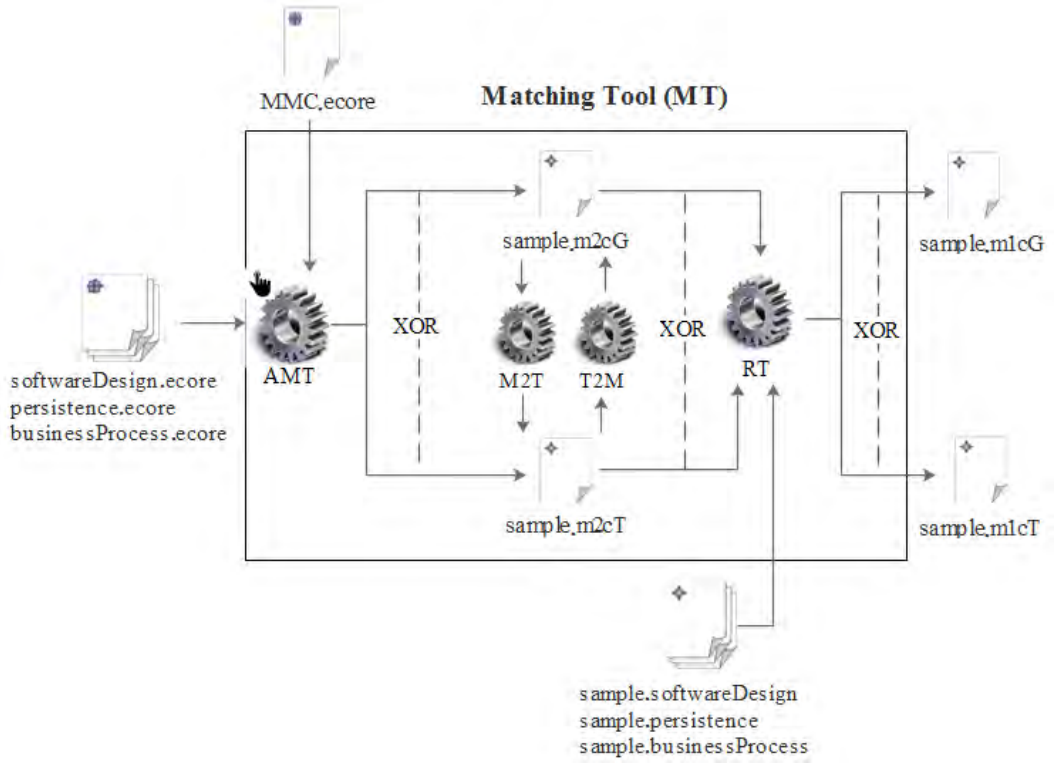


Figure VI-6 : Enchaînement fonctionnel du *Matching Tool*

VI.4.1.1. Module Assisted Matching Tool (AMT)

Le rôle du module AMT est de créer le modèle M2C à partir des métamodèles décrits en utilisant Eclipse Ecore. Ce modèle de correspondance peut être obtenu à partir d'un éditeur graphique ou bien d'un éditeur textuel synchronisé de façon bidirectionnelle.

Avant de commencer la création des correspondances, l'expert intégrateur a la possibilité d'étendre le métamodèle de correspondance afin d'ajouter de nouveaux types de relation. Un menu, commun aux deux éditeurs graphique et textuel propose une action «Extend MMC» qui redirige l'utilisateur vers une interface d'ajout d'un type de relation. La Figure VI-7 illustre l'ajout du type de relation *Contribution*.

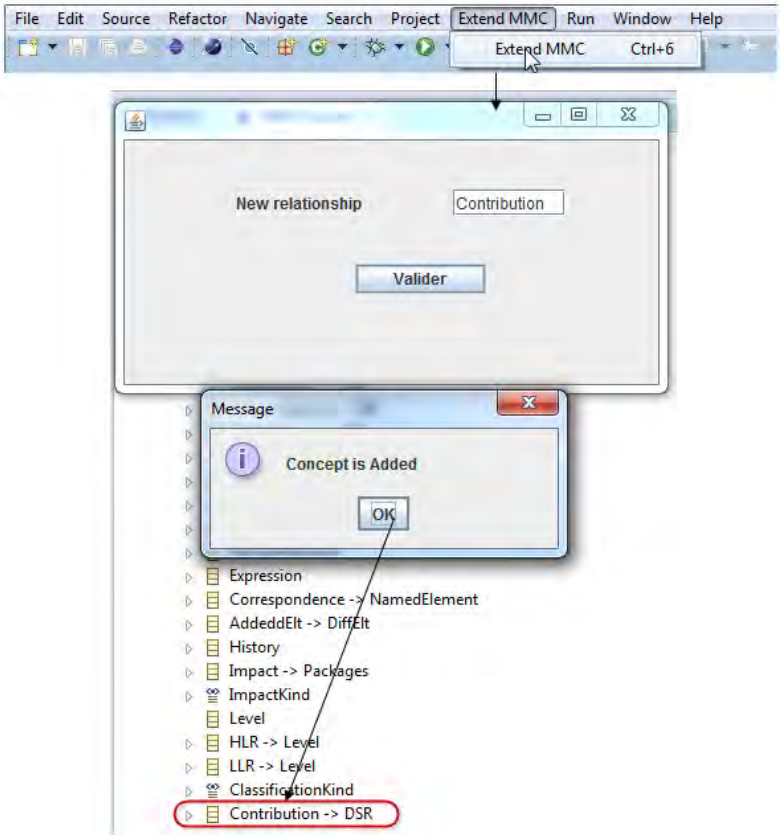


Figure VI-7 : Extension du MMC par un type de relation

VI.4.1.1.1. *Editeur graphique*

Le fonctionnement général de l'éditeur graphique est intuitif et sa prise en main rapide, pour tout utilisateur d'outil de modélisation. L'expert commence par choisir les métamodèles à relier (1). La liste des méta-éléments (instances de EClass de Ecore) du métamodèle sélectionné est chargée (2). L'expert désigne par la suite les méta-éléments participant à la correspondance (3). L'affichage des méta-éléments impliqués et les métamodèles auxquels ils appartiennent respecte le pattern P = «Méta-élément ε Métamodèle». L'étape finale consiste à sélectionner le type de relation (4). Quand tous les éléments sont là, la correspondance peut être créée via le bouton *Apply*. L'affichage d'une correspondance respecte le pattern : «TypeDeRelation (P(_P)*,P(_P)*)» (5).

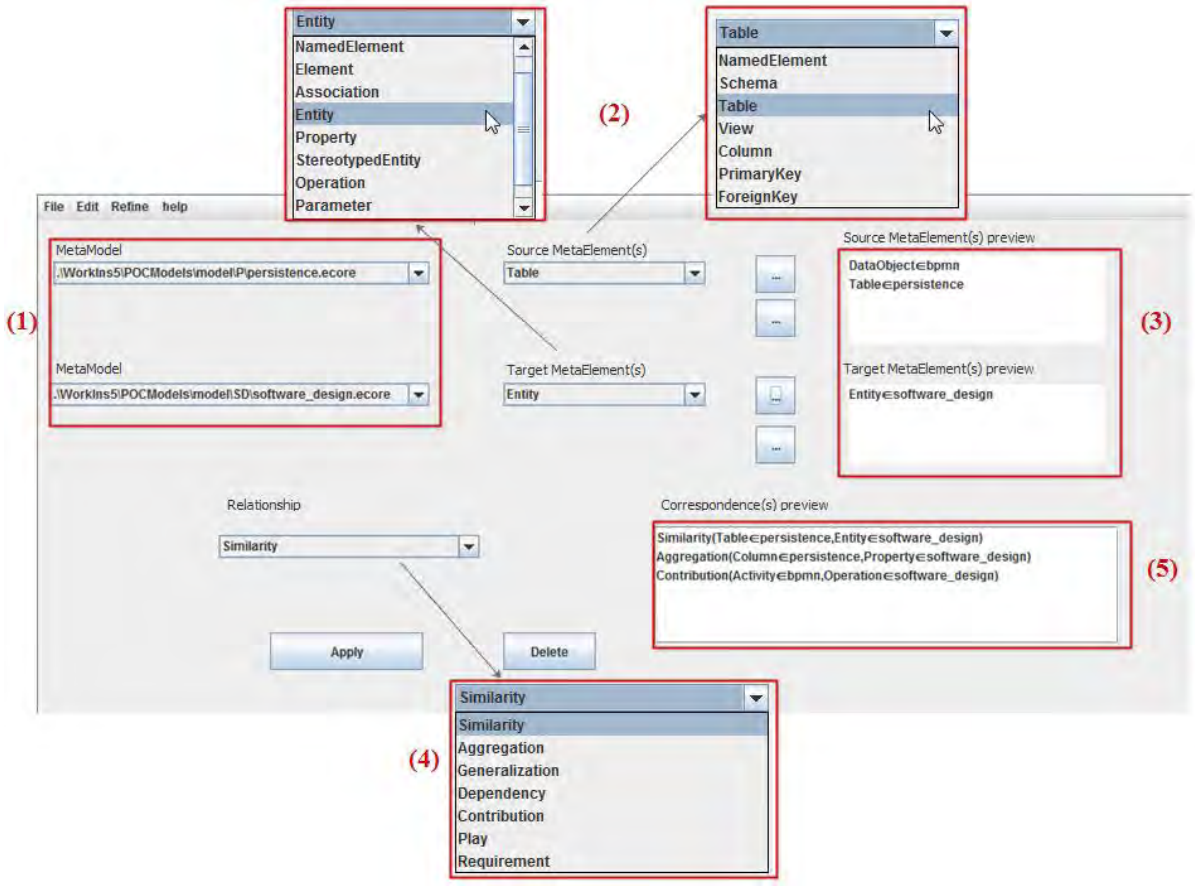


Figure VI-8 : Editeur graphique pour la création du modèle de correspondance M2C

Une fois que l'expert de domaine a créé l'ensemble des correspondances, il peut cliquer sur enregistrer, via le menu Fichier, ce qui engendrera la création du modèle de correspondance M2C.

VI.4.1.1.2. *Editeur textuel*

L'éditeur textuel, implémenté avec Xtext, permet aux personnes habituées à travailler en mode «édition source» de créer le modèle de correspondance. Pour cela l'expert, en s'appuyant sur la complétion automatique, coloration syntaxique, etc., commence par importer les (méta)modèles, puis crée le modèle de correspondance et définit les différentes correspondances.

La Figure ci-dessous montre un exemple du modèle de correspondance M2C du CMS sous l'éditeur textuel.

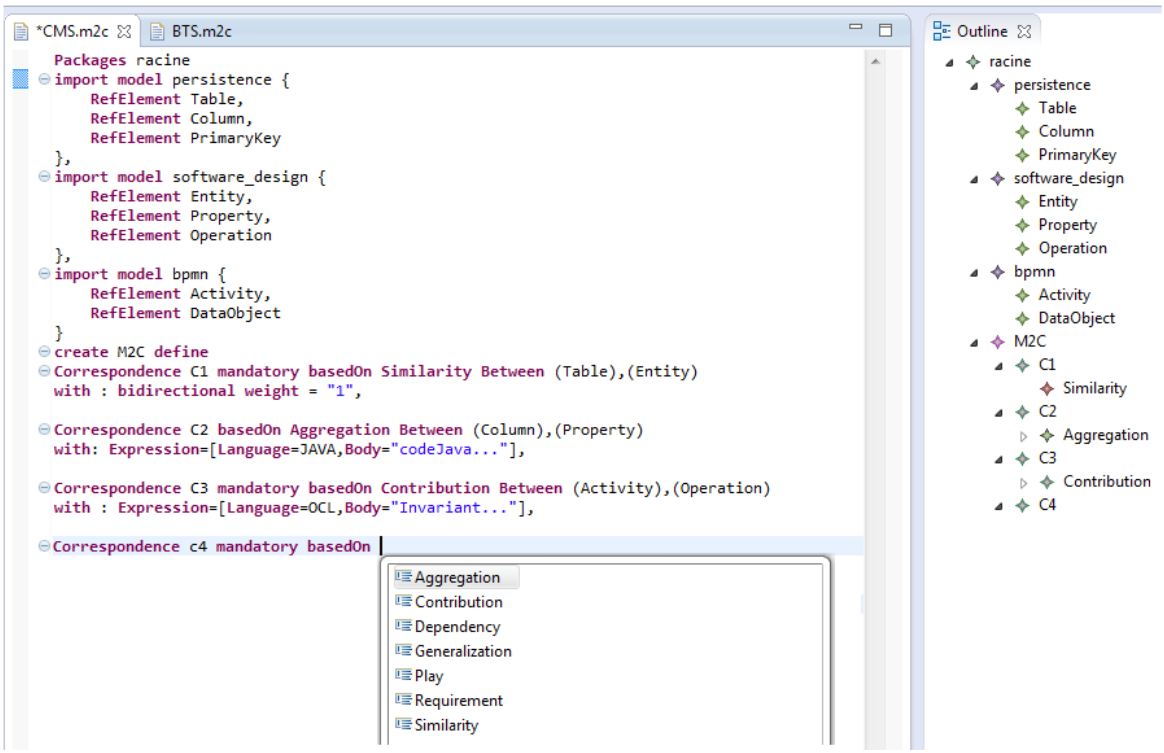


Figure VI-9 : Editeur textuel pour la création du modèle de correspondance M2C

La syntaxe de l'éditeur est basée sur une grammaire définie en EBNF (Annexe B). Le début de la grammaire permet d'importer le métamodèle Ecore du MMC. Grâce à cela, des références sont définies vers ce métamodèle afin de spécifier les concepts correspondant à chaque forme textuelle.

À titre d'exemple, la correspondance C3 de la Figure VI-9 définit un type de relation *Contribution* entre les méta-éléments *Operation* et *Activity*. Cette correspondance doit être maintenue en cas d'évolution de modèles, d'où le mot clé *mandatory,* et possède une expression sémantique écrite en langage OCL.

L'expert pourra à titre accessoire utiliser la vue *outline*. Elle permet d'avoir une vue hiérarchique sur les correspondances créées et d'accéder directement à une correspondance.

VI.4.1.2. Module Refining Tool (RT)

Le rôle de ce module (cf. Figure VI-10) est de produire le modèle M1C pour les modèles source en entrée. Les modèles source sont obtenus à partir des DSLs construits en exploitant les techniques de génération de code d'EMF. Pour chaque DSL, un modèle de générateur EMF (*EMF Generator Model – genmodel*) est créé à partir d'un métamodèle. Ce modèle sera exploité pour la génération du code source du modèle, des adaptateurs et de l'éditeur.

Le RT se base sur trois principaux modules : ReproT (*Reproduction Tool*), MFT (*Model Filtering Tool*) et OFT (*Ontology Filtering Tool*). Le premier permet de mettre en place la première étape du raffinement par propagation qui est la reproduction (voir section IV.7.3.1). L'Annexe A présente l'algorithme éléboré pour cette opération. Quant à l'opération de sélection (voir section IV.7.3.2), elle est mise place par les deux autres modules.

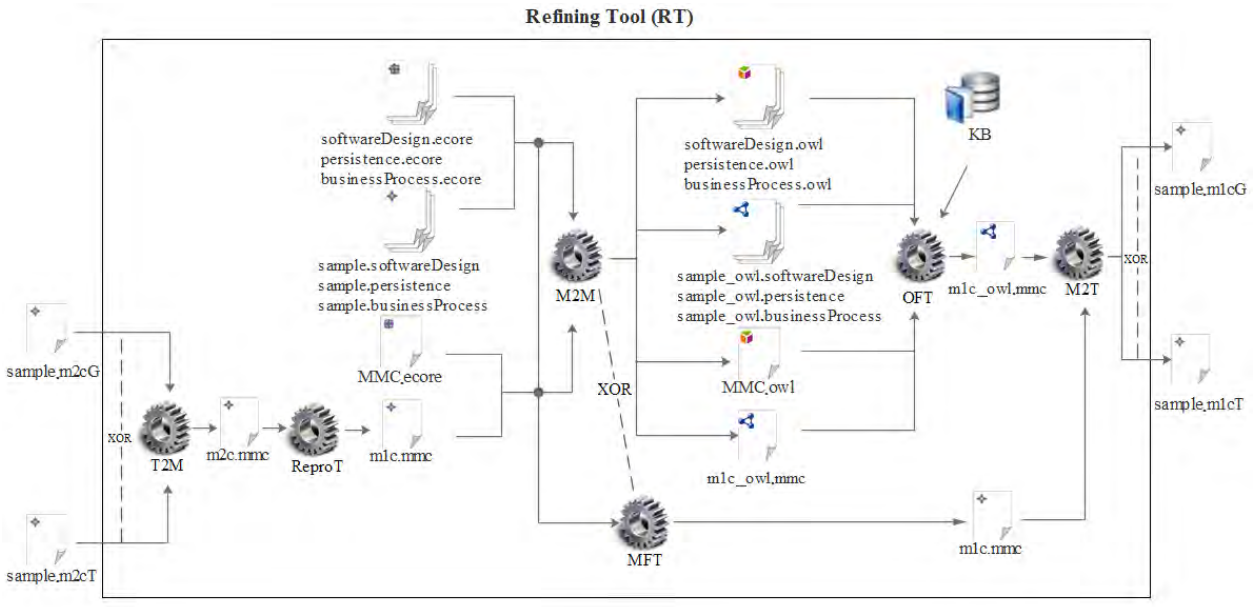


Figure VI-10 : Enchainement fonctionel du «Refining Tool»

Le but du module ReproT est d'éviter la création de correspondances entre les éléments ayant des méta-éléments qui ne participent pas dans une HLC. Le nombre de correspondances est ainsi réduit mais cela ne garantit pas que les correspondances générées soient toutes correctes.

Afin de combler ce manque, le module MFT a pour objectif de renforcer l'automatisation de la mise en correspondance en utilisant les expressions sémantiques des types de relation (voir section IV.5). On a recours au module OFT quand les expressions sémantiques reposent sur des fonctions décrites en utilisant une base de connaissances (*Konwledge Base*). Cette dernière est constituée d'un ensemble de métriques de mise en correspondance d'ontologies et d'une ontologie spécifique au domaine d'application traité (si elle existe).

Pour cela le module M2M opère une transformation de l'ensemble des modèles et de leurs métamodèles respectifs en OWL. Le processus de transformation est réalisé à l'aide du Framework TwoUse. Les métamodèles sont représentés comme des composants terminologiques (*Terminological component-TBox)* et les modèles comme des composants assertionnels (*Assertion component- ABox*).

Le RT intègre deux niveaux de raffinement : Propagation et Extension. Pour pouvoir utiliser le RT, il faut tout d'abord ajouter une vue appelée *Refining*. Cette vue contient l'ensemble des modèles M2C, créés via le module AMT, disponibles dans notre espace de travail.

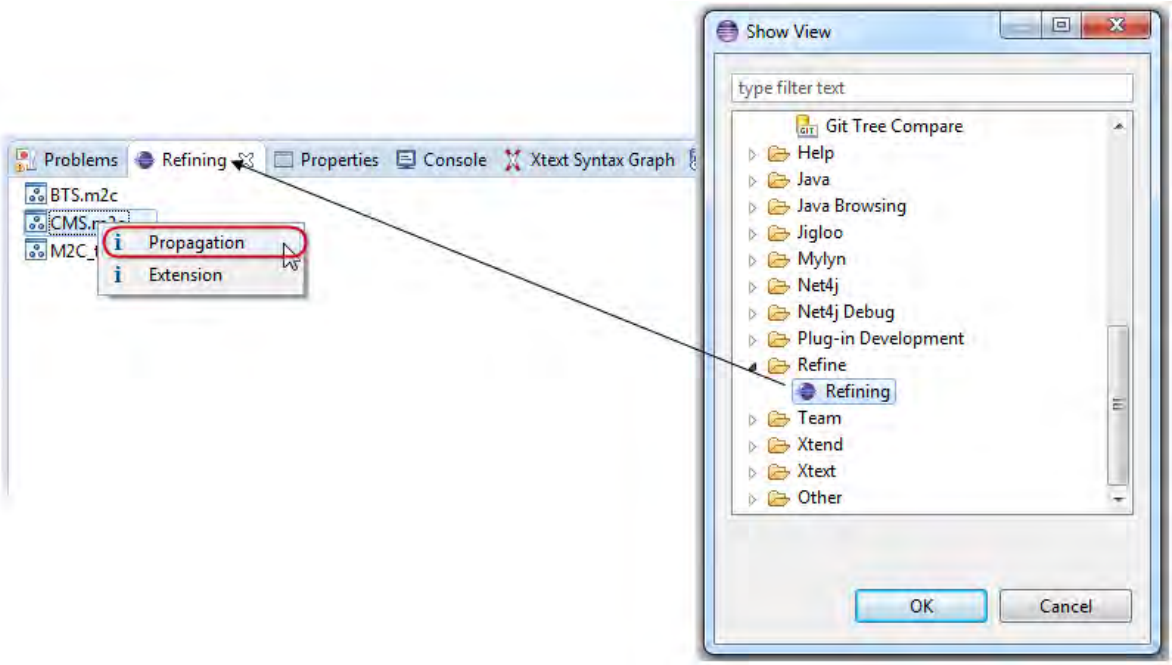


Figure VI-11 : Intégration de la vue *refining*

Comme décrit dans la section IV.7, le raffinement est une opération de transformation d'un modèle M2C en un modèle M1C. Cette transformation a été implémentée en Java/EMF et non pas avec un langage de transformation à l'instar de QVT ou ATL. En effet, le problème qui se posait est que la transition des HLCs vers les LLCs est non seulement horizontale mais aussi verticale. La transition est horizontale étant donné que c'est un passage d'un modèle M2C à un modèle M1C (comme illustré sur la Figure IV-16, les deux étant conformes au MMC), et elle est verticale au sens où les HLCs concernent les méta-éléments alors que les LLCs concernent les éléments du modèle. Le raffinement est donc une transformation qu'on pourra qualifier d'hybride. Par conséquent, cette transformation ne peut pas être mise en œuvre en utilisant des langages de transformation classiques puisqu'ils ne couvrent pas l'aspect hybride.

VI.4.1.2.1. ***Editeur graphique***

La Figure VI-12 illustre l'éditeur graphique du RT. Ce dernier permet d'afficher la liste des LLCs créées. Pour chaque LLC, le type de relation et les éléments de modèle associés sont affichés. Pour chaque élément de modèle on peut visualiser son type (méta-élément), le modèle auquel il appartient ainsi que le métamodèle auquel ce dernier est conforme.

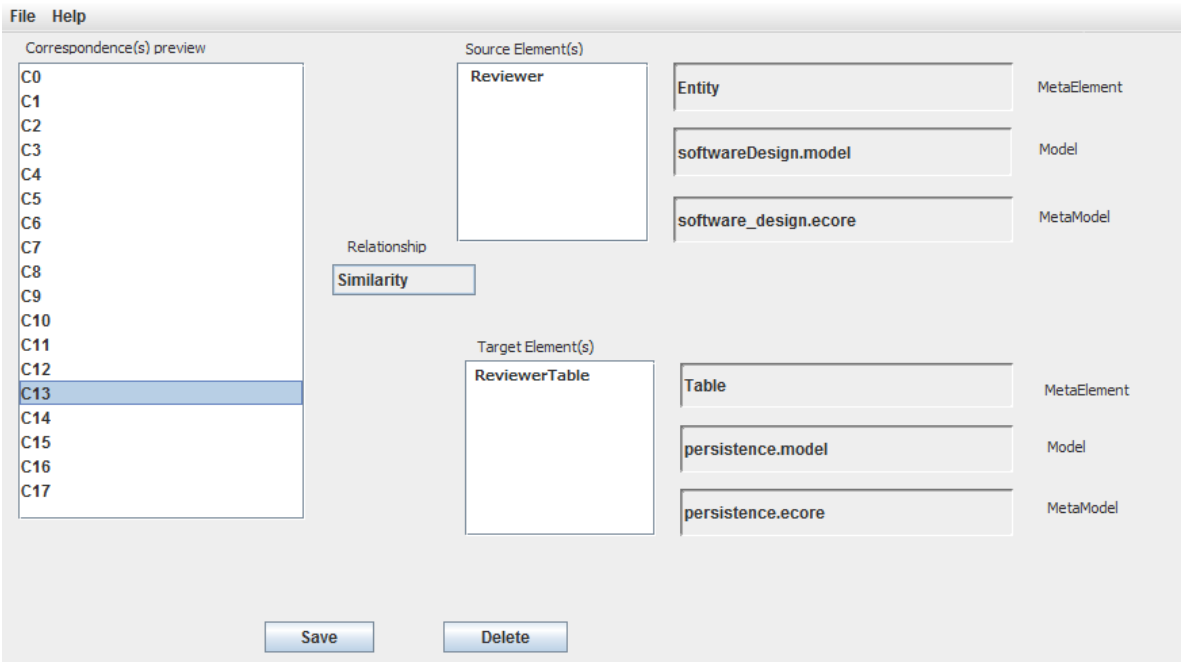


Figure VI-12 : Editeur graphique pour la création du modèle de correspondance M1C

VI.4.1.2.2. *Editeur Textuel*

L'éditeur Textuel du RT est semblable à celui du AMT (présenté plus haut). La seule différence est que cet éditeur contient des éléments de modèles au lieu des méta-éléments.

VI.4.2. Framework Consistency Management Tool (CMT)

L'implémentation de ce Framework commence par la mise en place du patron de conception observateur. Celui-ci permet d'écouter en temps réel les changements apportés aux différents modèles d'entrée et à les enregistrer sur le modèle M1C. Pour cela, comme présenté sur la Figure V-4 trois méthodes (notify(), attach(observer : Observer) et detach (observer : Observer)) doivent être ajoutées dans le code source java de l'éditeur de chaque DSL du CMS. Pour automatiser cet ajout nous avons modifié le processus de génération de code d'EMF en accédant aux templates JET et nous y avons greffé nos méthodes. Ainsi chaque DSL créé sera automatiquement mis à l'écoute.

Tous les changements sont gardés dans un fichier de trace. Par contre, le modèle de correspondance ne contient, en plus des ajouts, que les changements (de type modification et suppression) sur les éléments impliqués dans des correspondances.

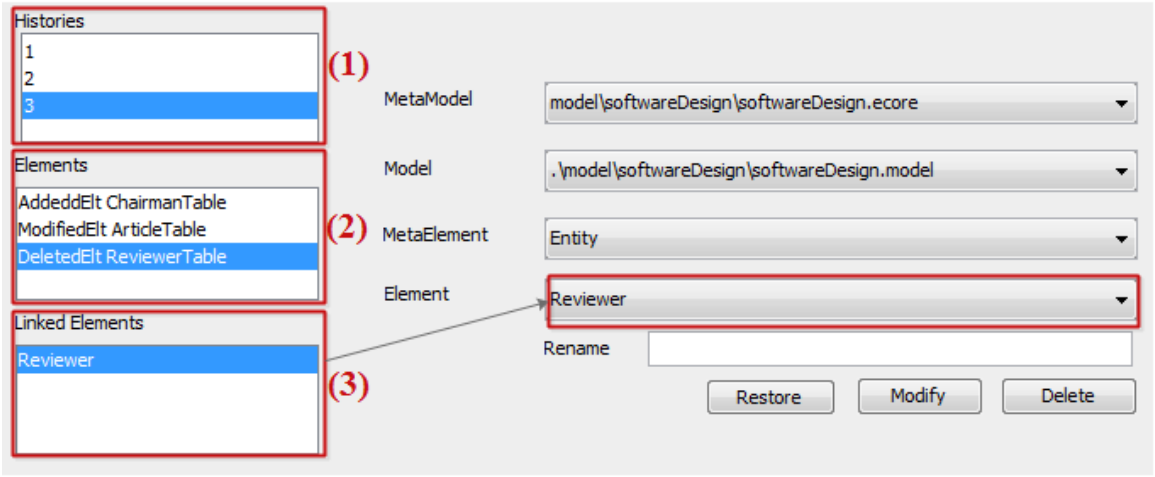


Figure VI-13 : Interface graphique du CMT

L'interface graphique (Figure VI-13) affiche l'historique de tous les changements (1). Pour la version numéro trois de l'historique, trois changements ont été effectués (2) à savoir : l'ajout d'un élément *ChairmanTable*, la modification de l'élément *ArticleTable* et la suppression de l'élément *ReviewerTable*. En double cliquant sur un de ces éléments, il est possible de connaître son type, le modèle auquel il appartient ainsi que le métamodèle. L'interface permet aussi de visualiser, pour un élément qui a subi un changement de type modification ou suppression, les éléments auxquels il est relié par une correspondance. Ces éléments sont susceptibles d'être influencés (3). Pareillement, les mêmes informations peuvent être visualisées en double cliquant sur l'un de ces derniers.

Trois actions permettent de mettre en oeuvre la décision de l'expert conformément au processus présenté dans la section V.8 du chapitre précédent. L'action *restore* permet de récupérer la version précédente de l'élément et l'action *modify* de le modifier. Si la correspondance n'est plus valide, l'action *delete* l'enlèvera du modèle de correspondance. Les changements opérés par ces trois actions sont eux aussi ajoutés à l'historique des changements.

VI.5. Conclusion

Nous avons présenté dans ce chapitre le prototype HMCS. Nous avons exhibé son architecture et les différents modules le constituant qui sont développés sous forme de plug-ins Eclipse. Nous avons aussi détaillé dans ce chapitre le scénario de fonctionnement de chacun d'entre eux.

Cette réalisation a profité des avantages offerts par l'approche MDE, notamment en termes d'automatisation de certaines tâches de développement. De plus, l'environnement Eclipse nous a permis de minimiser le temps nécessaire pour développer le prototype support de notre approche, comme par exemple la génération du code de base pour l'éditeur EMF et l'éditeur textuel.

Nous avons développé un outil générique qui prend en compte n'importe quel DSL à condition qu'il soit conforme au méta-métamodèle Ecore.

Les éditeurs graphiques que nous avons développés pour les modules AMT, RT et CMT proposent un format de représentation différent du format en 3 vues (2 vues pour les 2 modèles d'entrées et une vue pour le modèle de correspondance) proposé dans les approches telles que AMW et Modelink. Là où ce format ne permet pas de représenter plus que deux modèles, nos éditeurs permettent de représenter un nombre quelconque de modèles prêts à être utilisés en même temps.

La Figure VI-14 permet d'avoir une idée de notre prototype en termes de métriques de la qualité quantifiées à l'aide de CodePro Analytix (Google, 2010). Par exemple le nombre de lignes de code total est de 104 516 lignes effectives (nombre de lignes qui contiennent des caractères autres que des espaces blancs et des commentaires).

Metric	Value
⊞ Abstractness	16.1%
⊞ Average Block Depth	0.66
⊞ Average Cyclomatic Complexity	2.27
⊞ Average Lines Of Code Per Method	11.79
⊞ Average Number of Constructors Per Type	0.38
⊞ Average Number of Fields Per Type	14.33
⊞ Average Number of Methods Per Type	11.53
⊞ Average Number of Parameters	0.56
⊞ Comments Ratio	14.4%
⊞ Efferent Couplings	387
⊞ Lines of Code	104,516
⊞ Number of Characters	6,856,777
⊞ Number of Comments	15,096
⊞ Number of Constructors	248
⊞ Number of Fields	11,140
⊞ Number of Lines	176,073
⊞ Number of Methods	7,495
Number of Packages	150
⊞ Number of Semicolons	57,577
⊞ Number of Types	650
⊞ Weighted Methods	18,272

Figure VI-14 : Ensemble de métriques appliquées au HMCS

Le prototype développé est utilisé pour illustrer l'étude d'un service d'urgence d'un hôpital dans le chapitre suivant.

CHAPITRE VII. ETUDE DE CAS

"Even if you're the last person on Earth to see the light,

you never forget it"

— *Carl Sagan*

VII.1. Introduction

Ce chapitre est dédié à la validation de notre approche à travers une étude de cas mise en œuvre avec le prototype HMCS décrit dans le chapitre précédent. Cette étude de cas a pour objectif de modéliser le domaine d'application d'un service d'urgence. Le choix de cette étude est justifié par l'existence de nombreux acteurs ayant des métiers distincts susceptibles d'interagir sur ce système en exploitant des modèles hétérogènes qu'il faut coordonner.

Ce chapitre est organisé comme suit : nous présentons d'abord l'étude de cas, puis nous déroulons notre approche, en traitant pour commencer la mise en correspondance et ensuite le traitement de l'évolution des modèles. Nous concluons en résumant les principaux résultats.

VII.2. Description de l'étude de cas

L'informatisation au service de la société est de nos jours un enjeu fondamental de l'économie. De nombreux domaines stratégiques peuvent bénéficier de cette informatisation tels que l'éducation, l'environnement, la santé, la gestion de crises, le tourisme, etc.

Les services d'urgence (SUs) représentent des points névralgiques du système de santé de tout pays. Un système informatique permettant d'automatiser la gestion d'un service d'urgence est clairement un système complexe au même titre que ceux que nous avons présenté en introduction de cette thèse.

Ces services d'urgence doivent faire face à des situations parfois critiques (catastrophes naturelles, attentats terroristes, épidémies, etc.), lors desquelles la coordination entre les acteurs et la complémentarité des compétences sont essentiels. Cette pluridisciplinarité – complémentarité des points de vue – et la nécessaire coordination entre les acteurs doivent être prises en compte dès la conception de tels systèmes, de façon à ce que les différents modèles élaborés au cours de cette conception soient parfaitement synchronisés. La gestion non optimale des services d'urgence – que l'on peut constater assez souvent – vient en partie d'une prise en compte

insuffisante de ces facteurs lors de la conception. En outre, ces systèmes informatiques doivent être évolutifs car les lois, les règlements, les règles de gestion, les procédures opératoires, les contraintes de sécurité et de protection des données personnelles, etc., peuvent être amenés à changer.

Les modèles qui sont réalisés lors de la conception d'un système complexe doivent donc être reliés pour assurer la cohérence globale du système, et doivent aussi être réutilisables et évolutifs, permettant de tenir compte des inévitables changements. Dans la suite de ce chapitre, nous nous plaçons plus particulièrement dans le contexte du service d'urgence (SU) d'un hôpital public.

De nombreux acteurs interviennent dans le fonctionnement d'un SU, de l'aide soignante jusqu'au chirurgien. Pour modéliser un tel système, nous représentons le domaine d'application par trois domaines métier gérés séparément par les acteurs suivants :

- Architecte de procédé : responsable de la mise en place des protocoles à appliquer pour le personnel du SU ; il crée un modèle exprimé selon un métamodèle de processus métier,
- Ingénieur médical : responsable de la construction des maquettes numériques ; il crée un modèle, exploité par l'urgentiste, qui définit le formulaire pour la représentation du compte rendu de l'examen d'urgence (CRE),
- Informaticien : responsable de la représentation du système d'information ; il crée un modèle conforme au métamodèle de conception représentant l'organisation du SU.

VII.3. Modélisation du domaine d'application

Dans les sections suivantes, nous présentons les modèles produits par les acteurs présentés ci-dessus ainsi que les métamodèles auxquels ils sont conformes.

VII.3.1. Modèle de protocoles médicaux

Le métamodèle (Figure VII-1) utilisé pour modéliser le modèle de protocoles médicaux est celui de la notation BPM. Le métamodèle comprend, entre autres, les concepts de *lane*, *pool* et *task*. Un extrait du modèle de protocole médical (Figure VII-2) illustre différents protocoles à appliquer par chaque type de personnel. Ces protocoles sont des procédures précises et codifiées. Dans un SU, chaque type de personnel a un rôle précis.

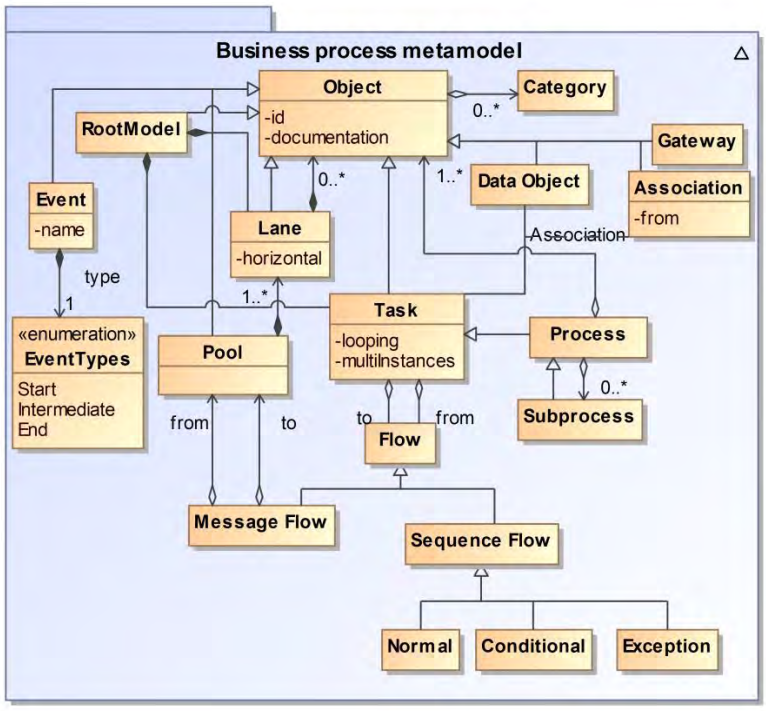


Figure VII-1 : Métamodèle de protocole médical

La Figure VII-2 représente un extrait de ce modèle. Tout d'abord, l'infirmière de tri accueille un patient et l'oriente vers une filière (médicale ou chirurgicale), puis l'aide-soignante installe le patient dans un box (ou chambre) et le prépare (notamment via un habillage médical). Ensuite, une infirmière applique un protocole interrogatoire au patient afin de récupérer des données administratives (exemple : numéro de sécurité sociale) et médicales (exemple : antécédents et allergies éventuelles). Le médecin urgentiste réalise alors une consultation et fait une prescription. L'infirmière retourne vers le patient et exécute la prescription du médecin qui peut comprendre des soins, des examens biologiques et/ou radiologiques, etc. Ultérieurement, le médecin urgentiste revoit le patient et, au vu des résultats de la prescription, effectue les actions suivantes :

- Poser un diagnostic (ex : AVC),
- Prescrire un traitement adapté au diagnostic,
- Orienter le patient vers :
  - Une hospitalisation (sur place),
  - Un transfert vers un autre hôpital, clinique, etc.,
  - Le retour à son domicile.
- Renseigner la cotation des actes réalisés, selon un certain barème et en utilisant un logiciel ad-hoc, en vue de codifier la facturation des différents actes (exemple : Consultation spécialiste : C2 ; Échographie transcutanée de la moelle épinière : AEQM001),
- Produire le compte-rendu d'examen d'urgence.

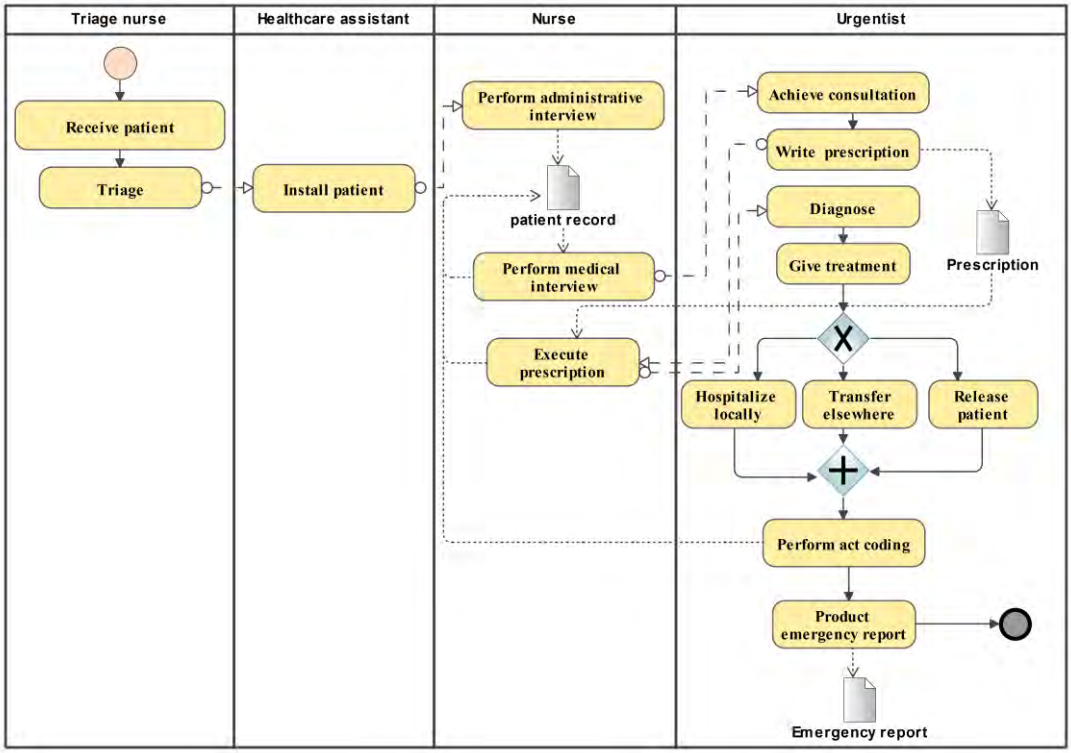


Figure VII-2 : Modèle du protocole médical à appliquer

VII.3.2. Modèle du Compte Rendu d'Examen (CRE) d'urgence

Le métamodèle proposé en Figure VII-3 permet la représentation d'un formulaire (*Form*) comme un ensemble de champs (*Field*) éventuellement composites.

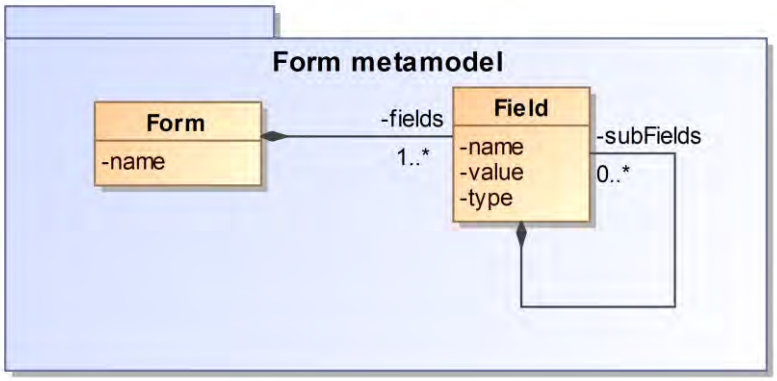


Figure VII-3 : Métamodèle de représentation d'un formulaire

Un compte rendu d'examen d'urgence (écrit ou dicté) est réalisé par le médecin urgentiste pour chaque patient au fur et à mesure des arrivées ; une observation clinique est rédigée dans le langage naturel utilisant des abréviations codifiées. Elle permet d'identifier la pathologie à traiter et d'orienter le patient vers le service adapté dans le même établissement ou dans un autre en fonction des informations listées ci-dessus ; quand il est informatisé, ce compte-rendu est rédigé

sur ordinateur via un formulaire représenté par la Figure VII-4 (selon la syntaxe concrète de HTML).

Le compte rendu d'examen (CRE) utilise les abréviations suivantes :

- MH : motif de l'hospitalisation,
- ATCDT : antécédents,
- TTTa : traitements antérieurs,
- ALL : allergies,
- HDM : histoire de la maladie,
- Ex clinique : examen clinique,
- Ex paraclinique : examens biologiques et/ou radiologiques,
- CCL°/CAT : conclusion/conduite à tenir (hospitalisation ou RAD (retour à domicile)),
- TTTs : traitements prescrits à la sortie.

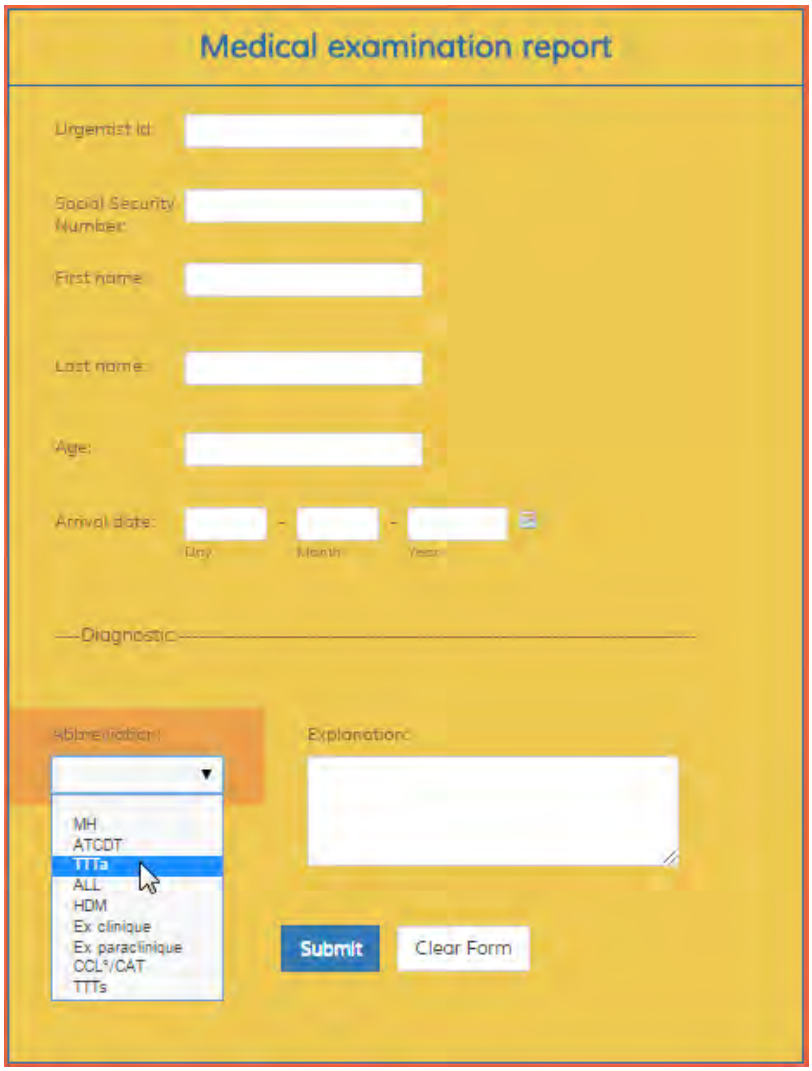


Figure VII-4 : Modèle du CRE d'urgence

VII.3.3. Modèle d'organisation du SU

Le métamodèle proposé (Figure VII-5) représente un aperçu du métamodèle de conception. Il s'agit d'un métamodèle adapté au paradigme objet, utilisé comme support à la conception. Le concept de base est celui de *Package*. Ce dernier contient les concepts *Class* et I*nterface*. Chacun d'eux peut contenir les concepts *Method* et *Attribute*.

Le modèle (Figure VII-6) conçu à partir du métamodèle de conception, permet de représenter, en utilisant la syntaxe concrète d'Ecore, l'organisation du SU en termes de personnels, dossiers médicaux et patients. Par exemple, un service d'urgence contient différents types de personnels. Pour chacun d'eux un planning de travail est établi. Un urgentiste est un personnel opérationnel qui soigne des patients. Pour chaque patient un diagnotic est effectué et des prescriptions sont généralement établies.

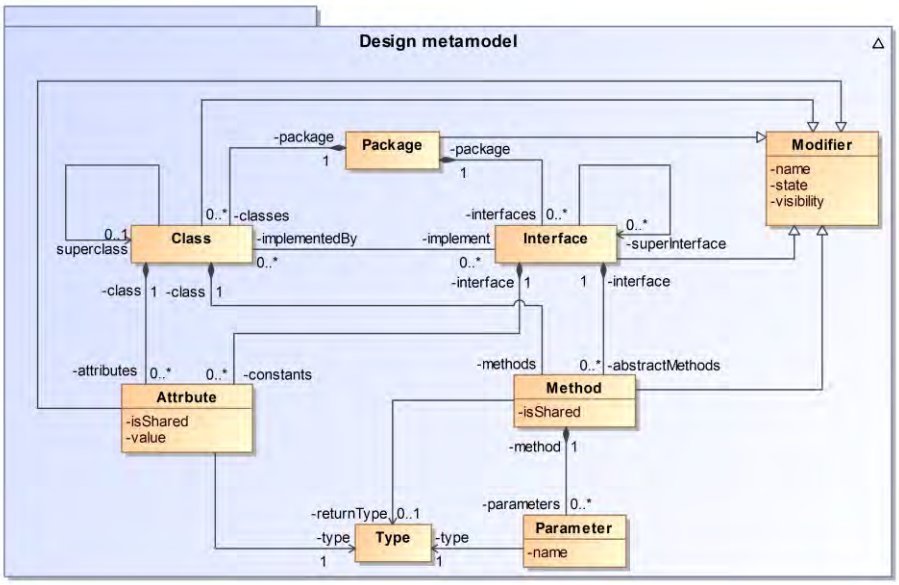


Figure VII-5 : Métamodèle de conception

Figure VII-6 : Modèle de conception de l'organisation du Service d'Urgence (SU)

VII.4. Application de l'approche au Service d'Urgence

VII.4.1. Mise en correspondance des modèles du SU

Comme nous l'avons présenté dans le Chapitre IV en section IV.4, le processus de mise en correspondance comporte cinq phases. Dans la suite, nous rappelons le rôle de ces phases et montrons leur application au SU.

VII.4.1.1. Phases 1 et 2 : Vérification de la complétude du MMC et extension du DSR

L'objectif de cette phase est de vérifier si les types de relations génériques définis dans le MMC sont suffisants pour établir les correspondances nécessaires entre les modèles du domaine d'application du SU. Compte tenu des spécificités de ce domaine, l'expert (intégrateur) ajoute les types de relations suivants : *Requirement, Deduction* et *Induction*. Le premier permet de connaitre les

champs dont une tâche a besoin pour son bon déroulement. Le deuxième permet de déduire, par le biais d'une fonction, la valeur d'un autre élément. Le troisième permet de représenter les opérations qu'une tâche invoque pour son exécution. Comme illustré sur la Figure VII-7, l'expert intégrateur peut ajouter les types de relations manquants au MMC, en utilisant l'outil HMCS.

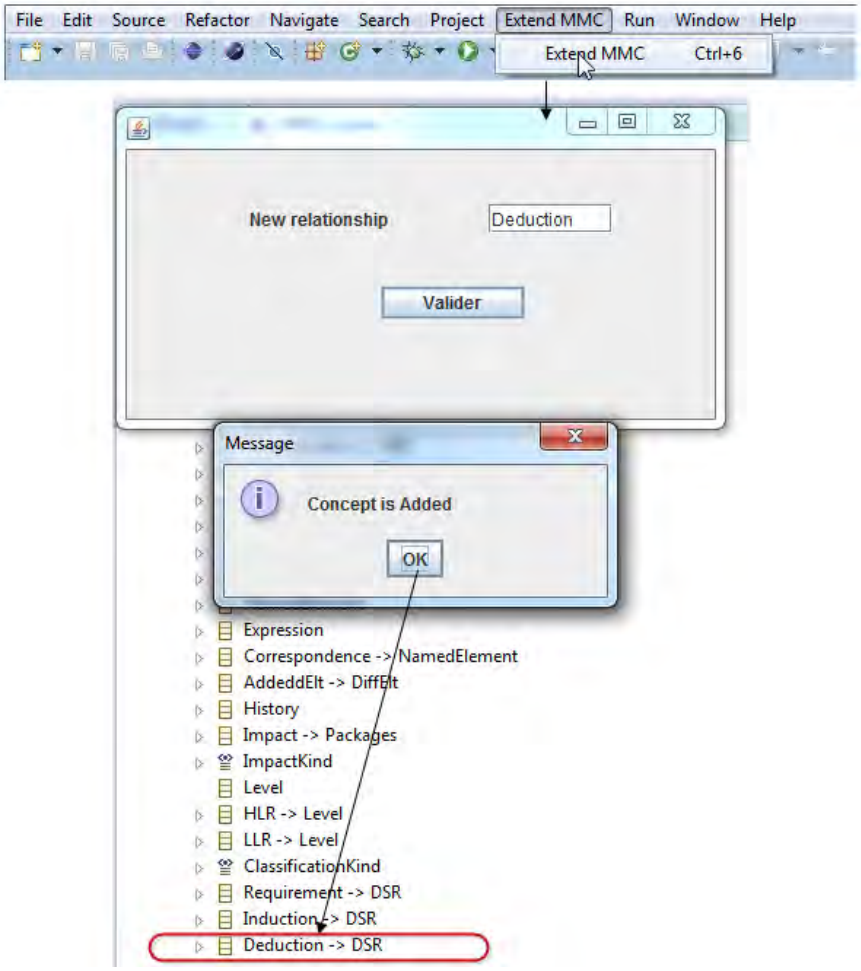


Figure VII-7 : Ajout de type de relation au MMC

VII.4.1.2. Phase 3 : Ajout d'une sémantique aux types de relation

Cette phase a pour rôle d'attribuer, via le DSL SED, une expression sémantique à chaque type de relation ajouté. L'expert associe des expressions sémantiques aux trois types de relations spécifiques ajoutés à savoir : *Requirement*, I*nduction*, *Deduction*.

La Figure VII-8 illustre le modèle d'expression de la sémantique (conforme au DSL SED). Dans le cas du SU, le type de relation *Requirement* est utilisé dans une correspondance impliquant deux éléments. Le premier est de type *Task* alors que le second est de type *Class*. *Requirement* est décrit par une expression en langage Java qui se base sur la méthode *sameAs* définie dans la section IV.7.3.2.2. Le type de relation *Deduction* est utilisé dans une correspondance impliquant un élément de type *Field* et un élément de type *Attribute*. Il est décrit par une expression qui se base sur la méthode *deduce* qui déduit un nombre à partir d'une date et inversement. Par exemple, l'âge à partir de la date de naissance (exemple qui sera illustré en section VII.4.1.4) ou bien le nombre

de jour d'hospitalisation aux urgences à partir de la date d'entrée, etc. Cette fonction se base sur l'API Joda-time[15]. Enfin le type de relation *Induction* se base sur la fonction *induce* appliquée aux éléments de type *Task* et *Operation*.

La définition des fonctions utilisées est la suivante :

- Deduce :

```
Public boolean deduce (LocalDate birthdate)
{
LocalDate now = new LocalDate();
//Calculate age from birthdate
Years age = Years.yearsBetween(birthdate, now);
if (this.equals((Integer)age).intValue())
return true;
return false;
}
```

- Induce :

```
Public boolean induce (String sentence)
{
if (this.contain(sentence) && (this.getParent().contain(sentence)))
return true;
return false;
}
```

[15] http://www.joda.org/joda-time/userguide.html

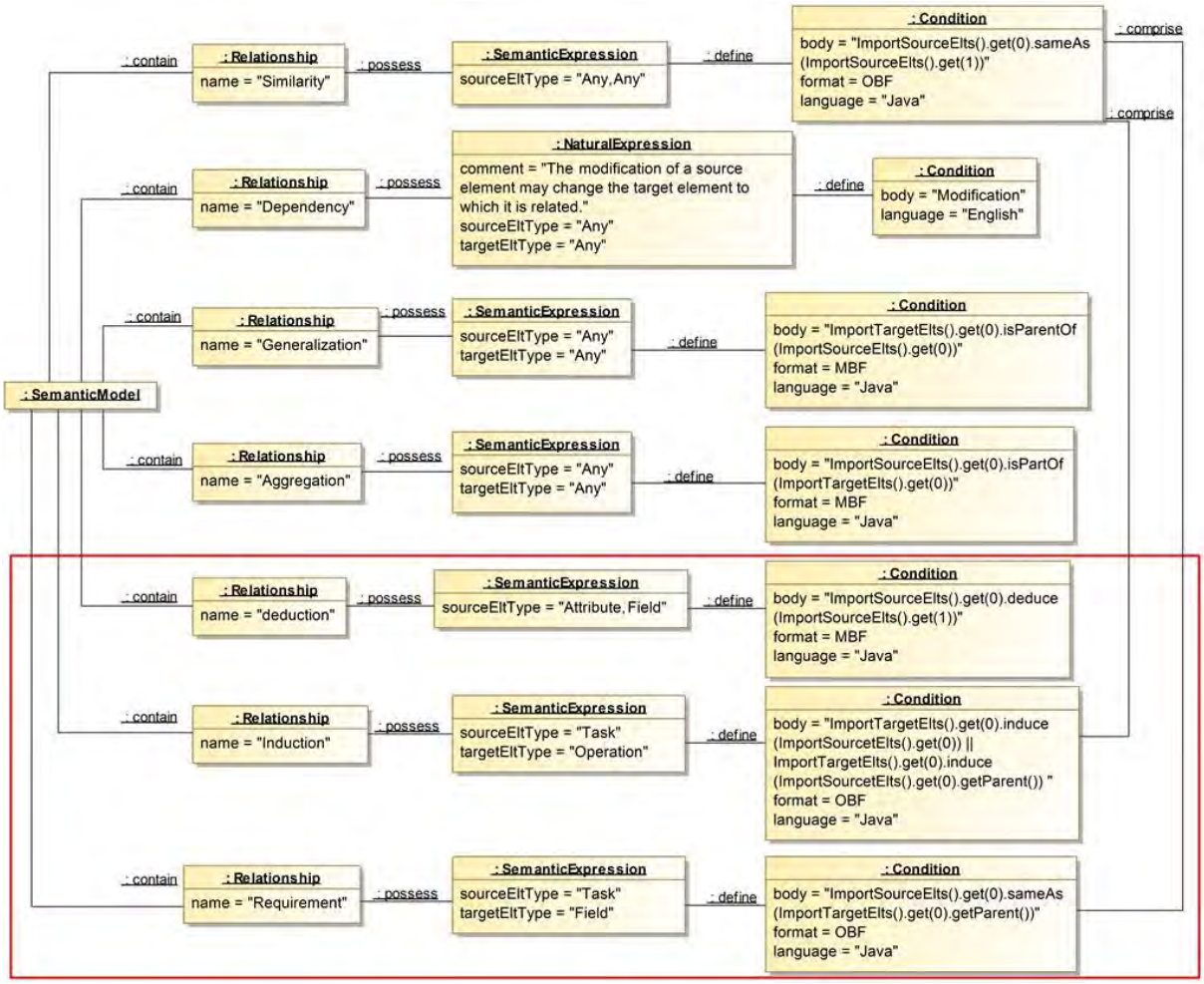


Figure VII-8 : Modèle d'expression sémantique pour le domaine du SU

Une fois le modèle d'expression de la sémantique créé, il est tissé au MMC pour donner le résultat illustré par la Figure VII-9.

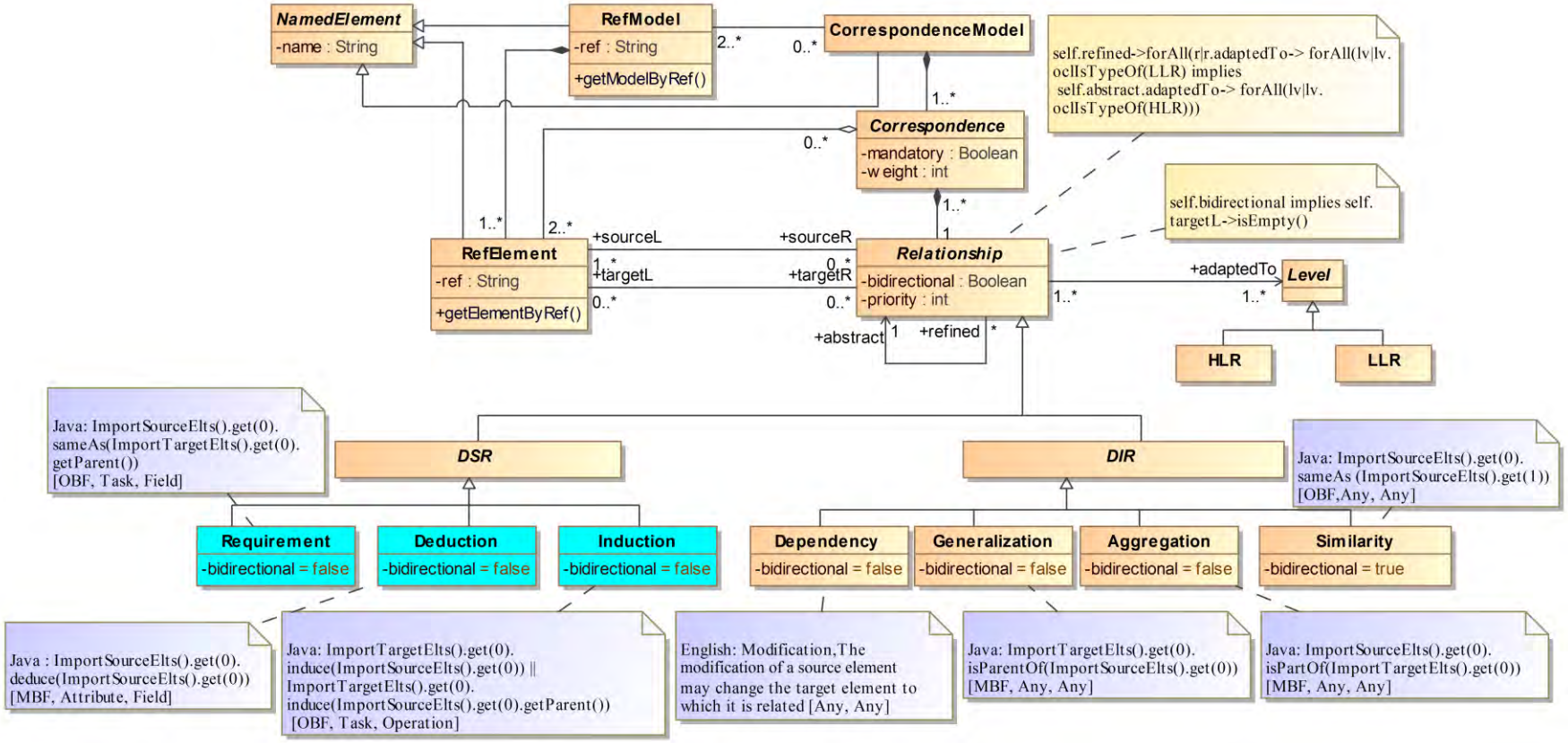


Figure VII-9 : MMC obtenu après tissage avec le modèle SE

VII.4.1.3. Phase 4 : Définition des correspondances au niveau M2

La Figure VII-10 détaille les correspondances établies entre les méta-éléments des métamodèles du domaine d'application. Le type de relation *Similarity* est défini entre les méta-éléments *Field* du métamodèle *Form* et *Attribute* du métamodèle *Design*. Ces deux méta-éléments sont aussi reliés par le type *Deduce*. Le type *Generalization* relie le méta-élément *Pool* avec le méta-élément *Class*. *Induce* est utilisé pour créer la correspondance qui relie les méta-éléments *Task* et *Operation*. Le dernier type *Requirement*, relie le méta-élément *Task* à *Field*. Mises à part les correspondances ayant le type de relation *Similarity* ou *Generalization*, créées manuellement par l'expert intégrateur, étant donné qu'ils ne sont pas spécifiques à un type précis (valeur *Any* dans l'attribut *sourceEltType* et/ou *targetEltType*), les autres correspondances sont créées automatiquement en fonction des valeurs spécifiées dans *sourceEltType* et *targetEltType* (cf. Figure VII-8).

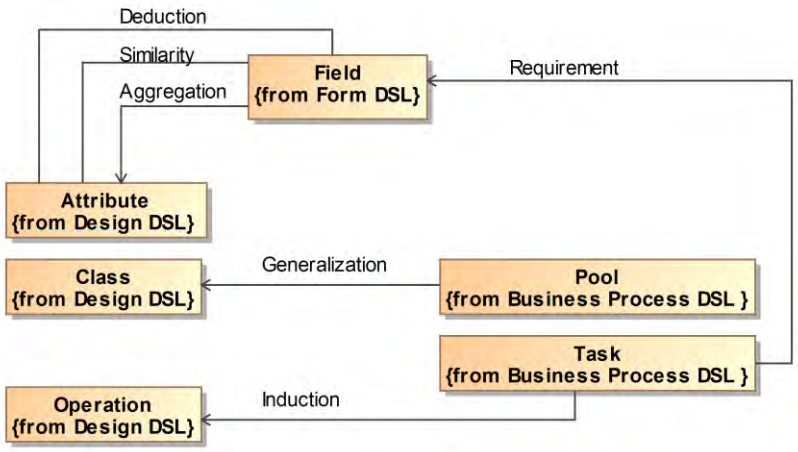


Figure VII-10 : Modèle M2C du domaine d'application SU

La Figure VII-11 est une représentation de l'équivalent de la Figure VII-10 dans le prototype HMCS. Nous avons choisi d'utiliser l'éditeur graphique sachant qu'il est aussi possible d'utiliser l'éditeur textuel (cf. VI.4.1.1.2). L'image instantanée indique que l'expert intégrateur, pour la correspondance en cours de création, a choisi le métamodèle *Design* et le métamodèle *Form*. Par la suite la liste de leurs méta-éléments a été chargée. L'expert a sélectionné les méta-éléments *Attribute* et *Field* appartenant respectivement aux métamodèles *Design* et *Form,* avec le type de relation *Aggregation*. Une fois les méta-éléments source et cible ainsi que le type de relation définis, la correspondance est créée.

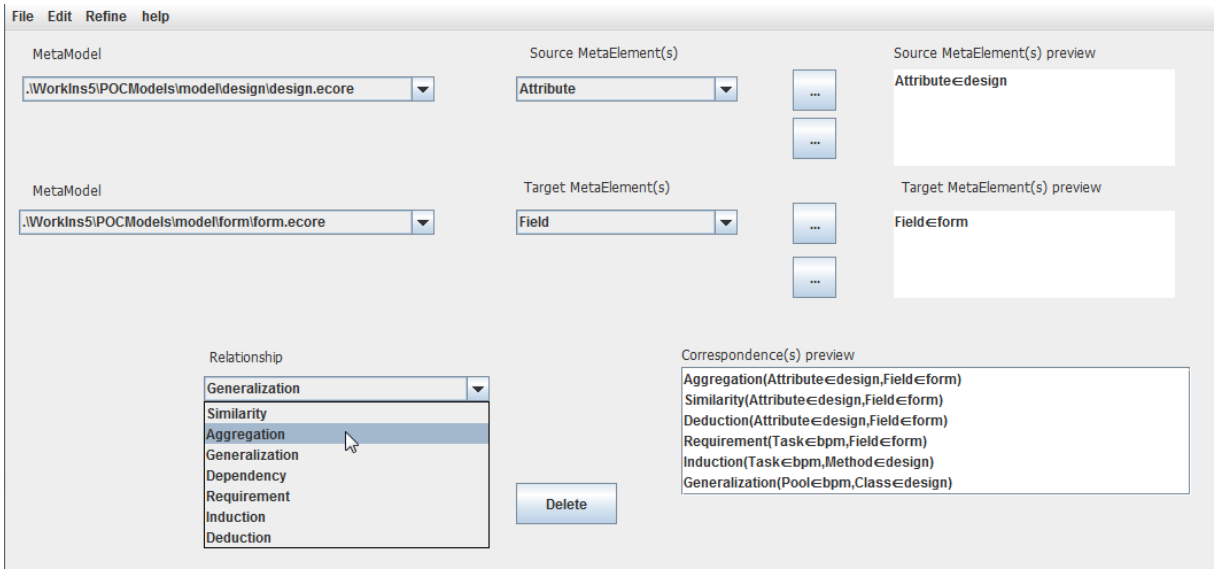


Figure VII-11 : Editeur graphique support à la création du modèle de correspondance M2C

À la fin de cette étape, l'expert attribue à chaque type de relation une valeur de priorité. Les valeurs de priorité : 1, 1, 2, 2, 3, 4, 5 sont attribuées respectivement aux types *Similarity*, *Dependency*, *Aggregation*, *Generalization*, *Deduction*, *Induction* et *Requirement*. L'utilisation de ces valeurs est présentée dans la section VII.4.2.3

VII.4.1.4. Phase 5 : Raffinement des correspondances au niveau M1

La création du M1C est d'abord réalisée via un raffinement par propagation des correspondances du M2C. Comme cela est décrit plus haut dans la section IV.7.3, ce type de raffinement consiste à exécuter une opération de reproduction suivie d'une opération de sélection. La première opération génère des LLCs entre les éléments dont le type participe à une HLC. La seconde opération permet de filtrer les LLCs précédemment créées en ne gardant que celles qui sont valides vis-à-vis des sémantiques définies. La Figure VII-12 montre les correspondances obtenues suite à l'opération de sélection.

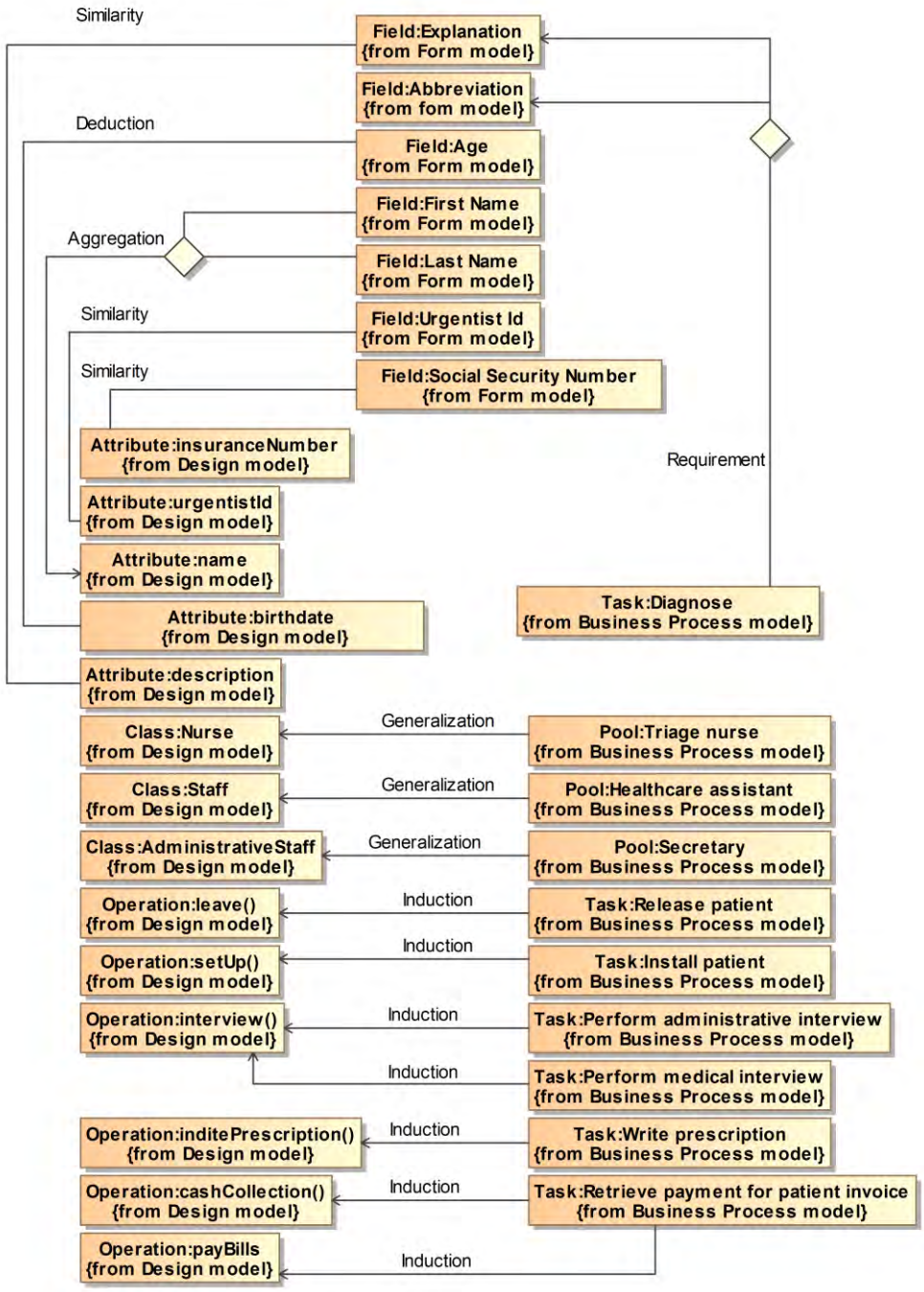


Figure VII-12 : Modèle M1C du SU obtenu par raffinement

La Figure VII-13 illustre les correspondances produites sous l'éditeur dédié à l'issue de la première opération (reproduction). Étant donné que l'implémentation de la seconde opération (sélection) est en cours de développement dans le prototype HMCS, nous avons ajouté un bouton *Delete* que l'expert intégrateur peut utiliser provisoirement afin de supprimer les correspondances non valides en regard des expressions préalablement définies. La Figure VII-14 et la Figure VII-15 montrent le M1C obtenu, après raffinement, avec respectivement les formats textuel et graphique.

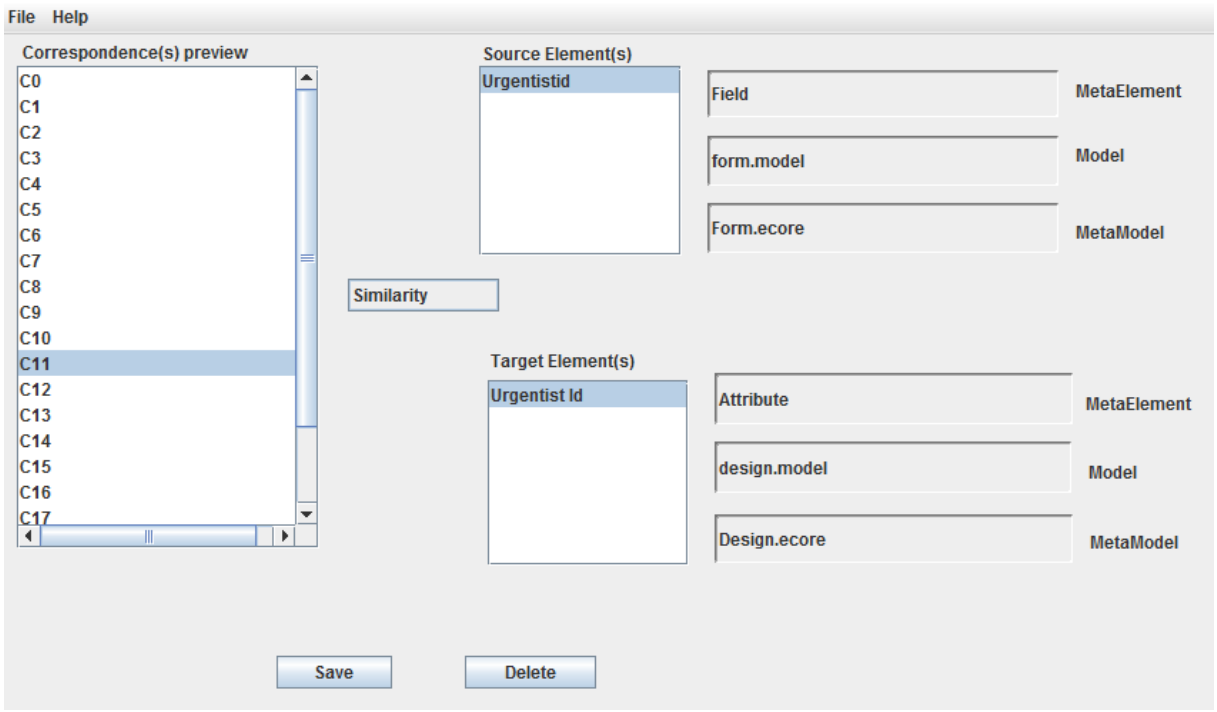


Figure VII-13 : Editeur graphique pour la création du modèle de correspondance M1C

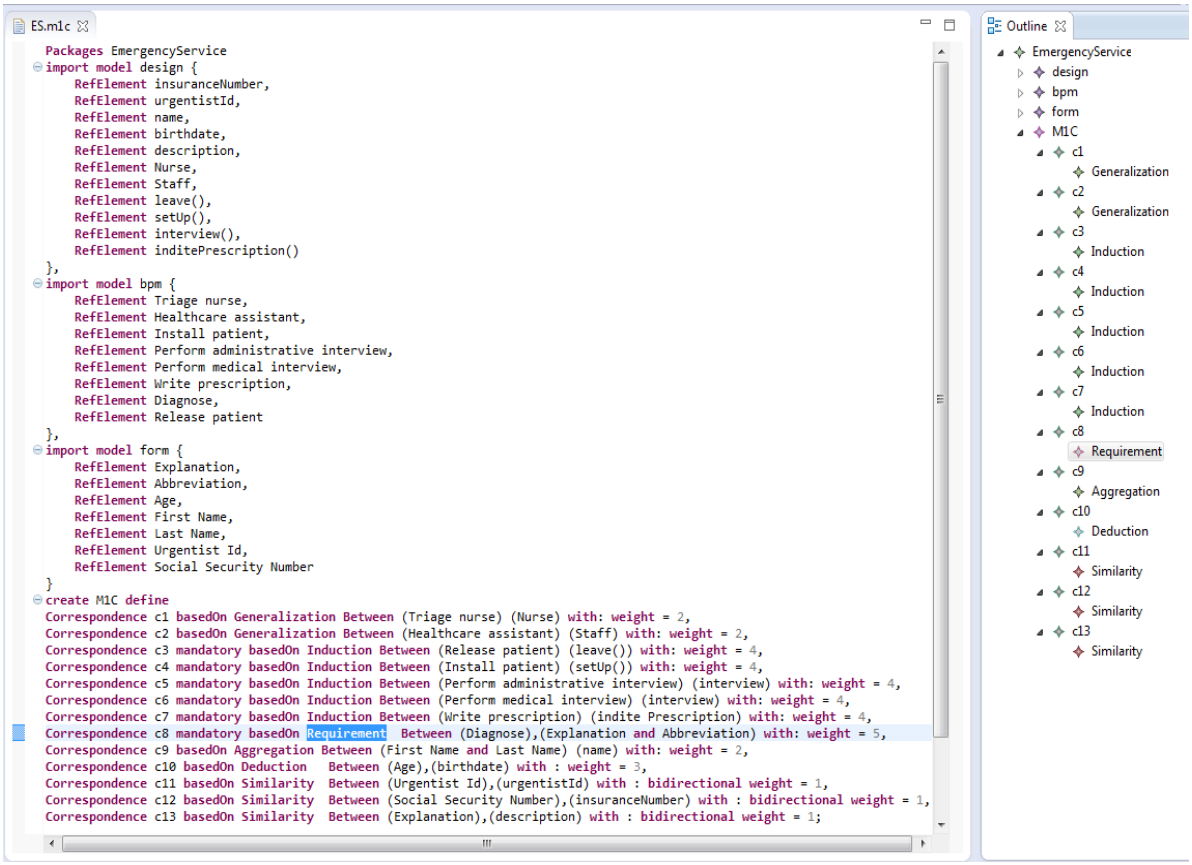


Figure VII-14 : M1C du SU obtenu par propagation (format textuel)

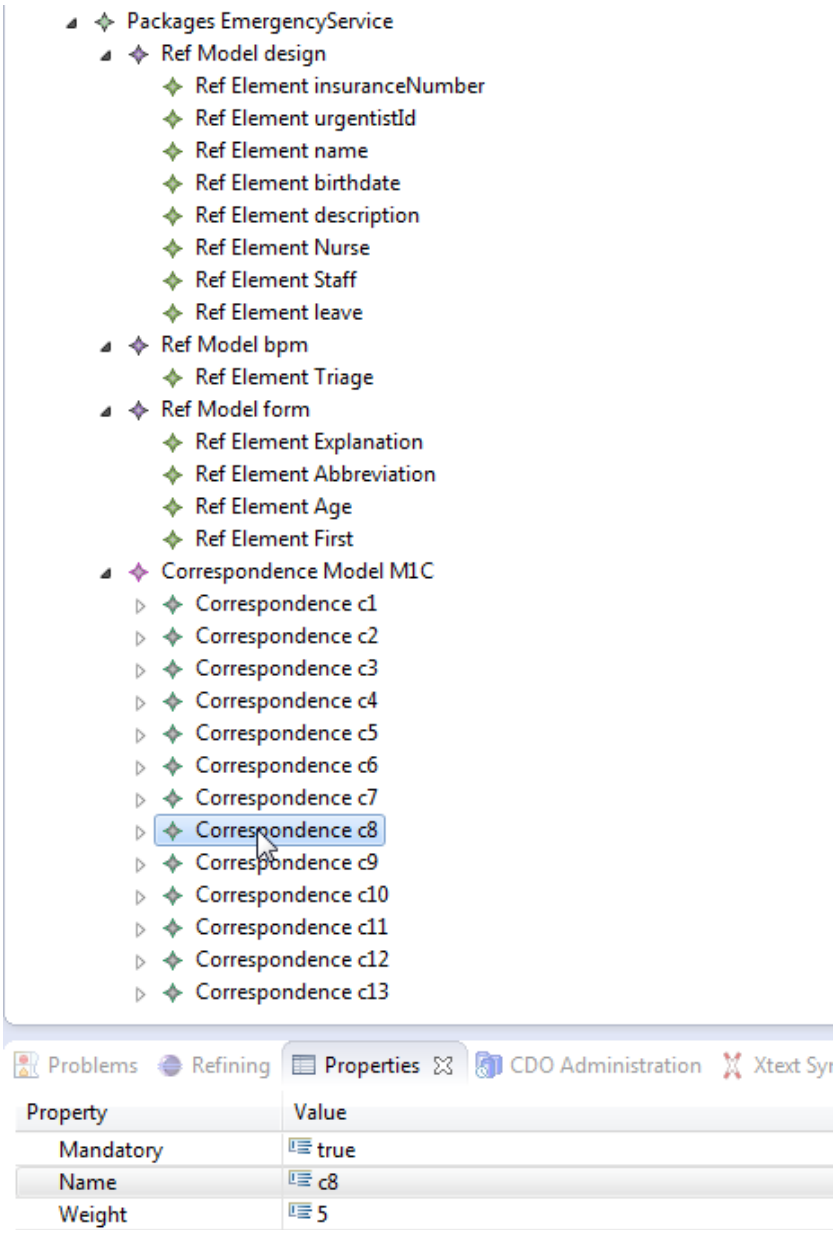


Figure VII-15 : M1C du SU obtenu par propagation (format graphique)

L'expert considère que certaines correspondances ayant le type de relation *Similarity* ou *Aggregation* ne sont pas suffisantes. Ces types sont remplacés respectivement par les types *Equality* et *Composition* via un raffinement par extension. La Figure VII-16 liste les correspondances concernées. Les éléments *Attribute:urgentistid* et *Field:Urgentist Id* décrivent le même concept et doivent donc contenir la même valeur d'où le type de relation *Equality*. Parallèlement, la correspondance ayant comme type de relation *Aggregation* reliant les éléments *Field:First Name* et *Field:Last Name* avec *Attribute:name*, est remplacée par une correspondance de type de relation *Composition*.

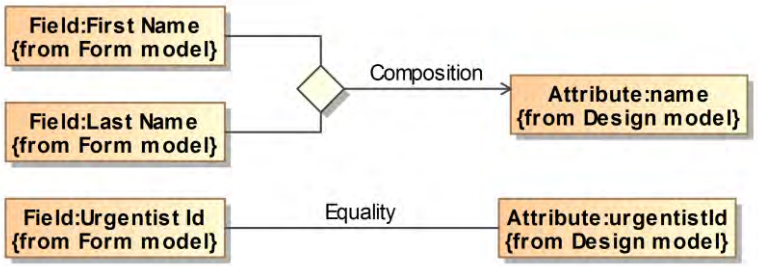


Figure VII-16 : Correspondances introduites suite au raffinement par extension

Dans la suite de ce chapitre nous nous focalisons sur les correspondances obtenues via un raffinement par propagation uniquement, donc les correspondances de la figure ci-dessus ne sont pas représentées dans le M1C utilisé dans la suite.

VII.4.2. Processus de maintien de la cohérence

Nous présentons dans cette section l'application au SU des six phases du processus de traitement de l'évolution que nous avons décrit en section V.2. Les changements traités dans la suite sont une simulation de ce qui peut se produire dans un SU.

VII.4.2.1. Phase 1 : Détection des changements

La partie 1 du Tableau VII-1 ci-dessous illustre les changements effectués sur les modèles du SU. Ces changements sont automatiquement détectés via l'outil HMCS et ajoutés au M1C, en fonction de leur type. Dans le modèle du processus métier (cf. Figure VII-17), supposons que soit ajouté le rôle secrétariat (*Secretary*). Il est responsable lors du départ du patient, d'imprimer les ordonnances ainsi que le compte rendu d'examen (CRE) d'urgence, puis d'éditer la facture et d'encaisser le paiement d'où l'ajout des éléments *Task:Print prescription*, *Task:Print emergency report*, *Task:Edit invoice* et *Task:Receive payment*. Au niveau du formulaire de CRE (cf. Figure VII-18), les éléments *Age* et *Arrival date* sont supprimés tandis que les éléments *Social Security Number* et *Explanation* sont respectivement remplacés par *Patient Record Number* et *Description*.

[1] Type de Changement [2] Ajout [3] Modification [4] Suppression [5] Mode de Classification [6] Automatique

Partie 1			**Partie 2**		
Type de Chgt.[1]	**Elément de modèle**		**Mode de Class.[5]**	**Elément(s) influencé(s)**	
				Influence directe	**Influence Indirecte**
A.[2]	*Pool:Secretary*		Auto.[6]	Aucune	Aucune
A.	*Task:Print prescription*		Auto.	Aucune	Aucune
A.	*Task:Print emergency report*		Auto.	Aucune	Aucune
A	*Task:Edit invoice*		Auto.	Aucune	Aucune
A.	*Task:Receive payment*		Auto.	Aucune	Aucune
S.[4]	*Field:Age*		Auto.	*Atttribute:birthdate*	Aucune
S.	*Field:Arrival date*		Auto.	Aucune	Aucune
M.[3]	Ancien	*Field:Social Security Number*	Guidé	*Attribute:insuranceNumber*	Aucune
	Nouveau	*Field:Patient Record Number*			
M.	Ancien	*Field:Explanation*	Guidé	*Task:Diagnose,* *Attribute:description*	Aucune
	Nouveau	*Field:Description*			

Tableau VII-1 : Aperçu des changements appliqués sur les modèles du SU, et des éléments susceptibles d'être influencés

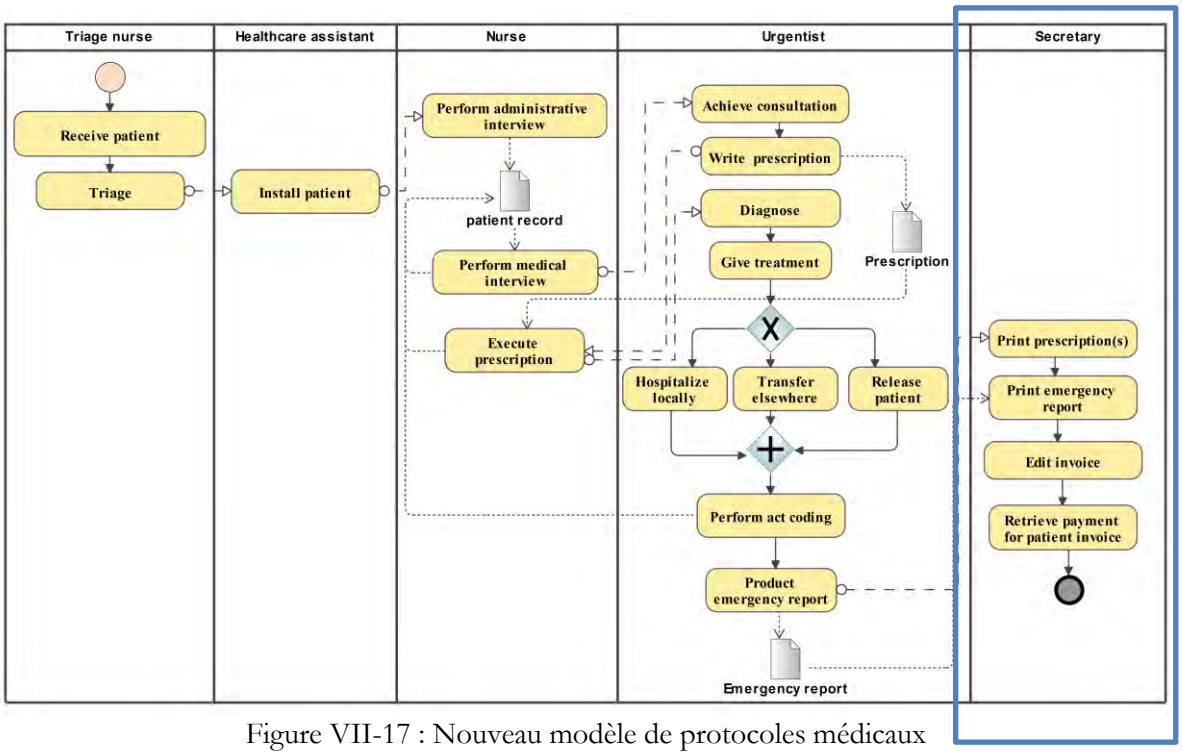


Figure VII-17 : Nouveau modèle de protocoles médicaux (ajout de la colonne de droite)

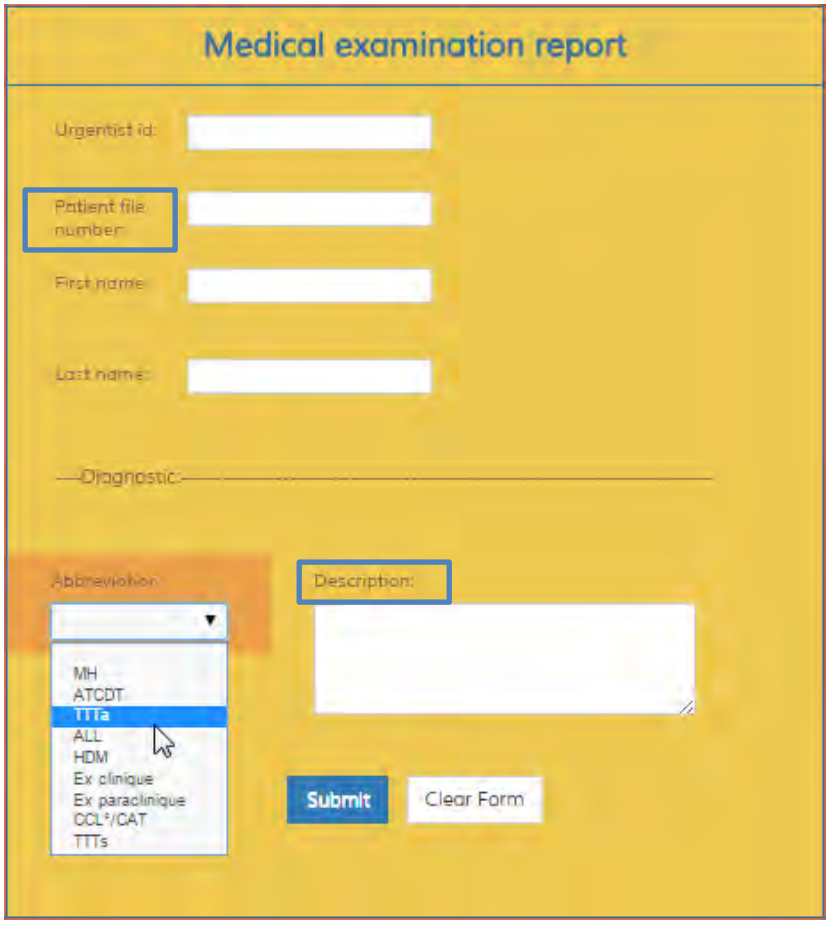


Figure VII-18 : Nouveau modèle du CRE d'urgence

VII.4.2.2. Phases 2 et 3 : Analyse des changements et gestion des cycles

La partie 2 du Tableau VII-1 permet de définir, pour chaque changement, à la fois le mode de classification (automatique ou guidé) et les éléments susceptibles d'être affectés, directement ou indirectement. Ces informations sont intégrées automatiquement dans le M1C par le HMCS. Par exemple, la suppression de l'élément *Field:Age* a une influence directe sur l'élément *Atttribute:birthdate* alors que la suppression de l'élément *Field:Arrival date* n'a pas d'influence sur les autres éléments de modèle. Ces deux changements sont classés en mode automatique. Les éléments de modèle ajoutés (par exemple : *Task:Print prescription, Task:print emergency report*, etc.) sont classés aussi en mode automatique et le fait de les ajouter n'a aucune incidence sur la cohérence du M1C existant. Les éléments concernés par les modifications sont classés en mode guidé ; la modification de l'élément *Field:Social Security Number* a une influence directe sur l'élément *Attribute:insuranceNumber*. Parallèlement, la modification de l'élément *Field:Explanation* a une influence directe sur les éléments *Task:Diagnose* et *Attribute:details*.

Concernant la gestion des cycles, l'outil HMCS n'a détecté la présence d'aucun cycle, ce qui est effectivement le cas.

VII.4.2.3. Phases 4 et 5 : Définition de la stratégie et priorisation des changements

L'expert intégrateur doit choisir entre deux stratégies : *Classification based Strategy* ou bien *Impact Based Strategy*. Dans notre cas, nous avons choisi la stratégie à base de classification en commençant par le mode guidé suivi du mode automatique. Ce choix est justifié par le besoin de traiter les changements de type modification (catégorisés en mode guidé) en premier avant de traiter ceux de type suppression ou ajout (catégorisés en mode automatique). Cela a donné lieu à la liste générée dans la partie 1 du Tableau VII-2.

Dans notre approche, nous traitons l'évolution d'un seul changement à la fois. Donc pour les changements classés en mode guidé par exemple, il faut choisir leur ordre de traitement. Pour cela nous calculons automatiquement un coefficient de pondération pour chaque changement (cf. section V.7 dans le Chapitre V).

Le résultat de la priorisation est présenté sous forme d'une nouvelle liste ordonnée en fonction des coefficients de pondération calculés (partie 2 du Tableau VII-2). Les changements sont traités dans l'ordre qui suit :

1. *Field:Description (weight=6),*
2. *Field:Social Security Number (weight=1),*
3. *Field:Age (weight=3),*
4. *Pool:Secretary (weight=0),*
5. *Task:Print prescription (weight=0),*
6. *Task:Print emergency report (weight=0),*
7. *Task:Edit invoice (weight=0),*
8. *Task:Receive payment (weight=0).*

L'ordre attribué aux changements de type ajout est aléatoire étant donné qu'ils sont traités à la fin du processus de traitement des changements (cf. Figure V-7) lors de la mise en correspondance.

[1] Type de Changement [2] Ajout [3] Modification [4] Suppression [5] Mode de Classification [6] Automatique

Partie 1			**Partie 2**		
Type de Chgt.[1]	**Elément de modèle**		**Mode de Class.[5]**	**Coefficient de pondération**	**Ordre**
M.[3]	Ancien	*Field:Social Security Number*	Guidé	Weight = Attribute:insuranceNumber * prioritySimilarity = 1	2
	Nouveau	*Field:Patient Record Number*			
M.	Ancien	*Field:Explanation*	Guidé	Weight = Task:Diagnose * priorityRequirement + Attribute:Details * prioritySimilarity = 6	1
	Nouveau	*Field:Description*			
A.[2]	*Pool:Secretary*		Auto.[6]	0	4
A.	*Task:Print prescription*		Auto.	0	5
A.	*Task:Print emergency report*		Auto.	0	6
A	*Task:Edit invoice*		Auto.	0	7
A.	*Task:Receive payment*		Auto.	0	8
S.[4]	*Field:Age*		Auto.	Weight = Attribute:name * priorityDeduction = 3	3

Tableau VII-2 : Ordonnancement des changements et calcul de la priorité dans le cas du SU

VII.4.2.4. Phase 6 : Traitement des changements

L'objectif de cette phase est de traiter les changements selon l'ordonnancement réalisé dans la phase précédente. Le Tableau VII-3 illustre les actions appliquées pour chaque changement identifié. Pour les deux premiers changements (*Field:Social Security et Field:Explanation)*, les correspondances concernées sont maintenues étant donné qu'elles restent valides après leur modification. Cette validité est vérifiée en exécutant le corps de la condition associé au type de

relation impliqué dans la correspondance (cf. section VII.4.1.2). La suppression du troisième élément (*Field:Age),* entraine la suppression de la correspondance puisqu'elle n'est pas obligatoire (*mandatory = false*) et qu'elle est devenue orpheline. En ce qui concerne le reste des changements de type ajout, l'opération de mise en correspondance est invoquée à la fin du processus pour chercher l'existence d'éventuelles correspondances et pour les créer par la suite.

[1] Type de Changement [2] Ajout [3] Modification [4] Suppression [5] Mode de Classification [6] Automatique

Ordre	**Type de Chgt.[1]**	**Elément de modèle**		**Mode de Class.[5]**	**Action effectuée**
1	M.[3]	Ancien	*Field:Social Security Number*	Guidé	Maintien de la correspondance
		Nouveau	*Field:Patient Record Number*		
2	M.[4]	Ancien	*Field:Explanation*	Guidé	Maintien de la correspondance
		Nouveau	*Field:Description*		
3	S.[3]	*Field:Age*		Auto.[6]	Suppression de la correspondance
4	A.[2]	*Pool:Secretary*		Auto.	Mise en correspondance
5	A.	*Task:Print prescription*		Auto.	Mise en correspondance
6	A.	*Task:Print emergency report*		Auto.	Mise en correspondance
7	A	*Task:Edit invoice*		Auto.	Mise en correspondance
8	A.	*Task:Receive payment*		Auto.	Mise en correspondance
9	A.	*Pool:Secretary*		Auto.[6]	Mise en correspondance

Tableau VII-3 : Actions effectuées pour chaque changement

Au niveau du HMCS, la Figure VII-19 illustre les fonctionnalités apportées pour gérer les traitements des changements. On affiche pour chaque historique (encadré 1) la liste des changements (encadré 2) ordonnée à l'aide des coefficients de pondération calculés (cf. section VII.4.2.3). Pour le deuxième changement (*Social Security Number*) à traiter, la liste des éléments susceptibles d'être impactés est affichée dans l'encadré 3. Comme nous l'avons précisé précédemment (section VI.5), l'implémentation de la sémantique permettant d'exécuter les corps de la condition associés aux types de relation n'est pas encore terminée. Donc, pour les changements de type modification, nous avons ajouté l'encadré 4 qui permet une fois l'expert assuré de la validité d'une correspondance, de décider, en fonction du processus présenté dans la Figure V-7, de l'action à effectuer. L'action *restore* permet de récupérer la version précédente de

l'élément et l'action *modify* de modifier l'élément de l'extrémité de la correspondance. Si la correspondance n'est plus valide, l'action *delete* l'enlève du modèle de correspondance.

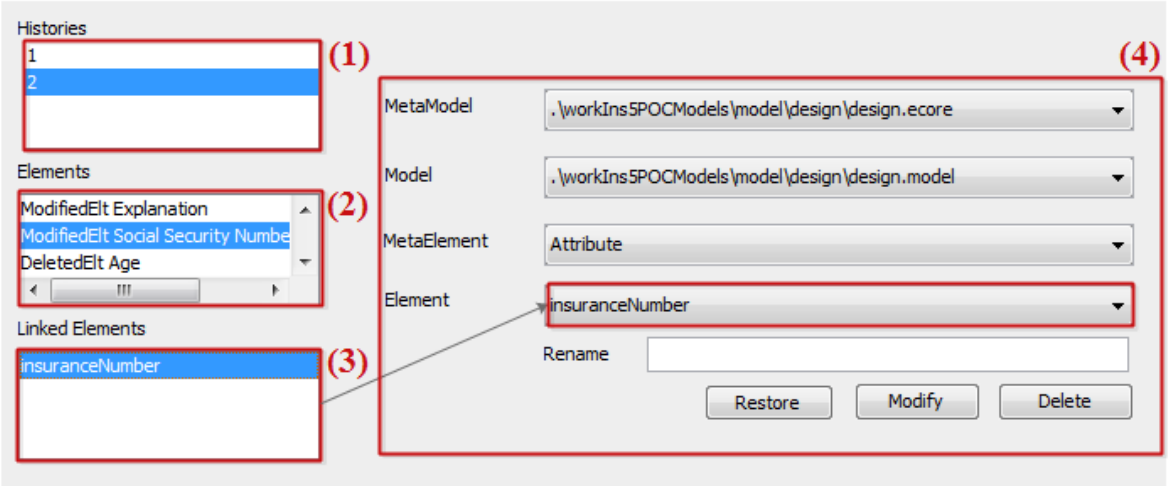


Figure VII-19 : Interface de traitement de l'évolution de modèles

La Figure VII-20 illustre le nouveau M1C du SU obtenu après le traitement des changements. Les correspondances C11 et C12 ont été modifiées alors que les correspondances C13, C14 et C15 ont été ajoutées.

ES.m1c

```
Packages EmergencyService
import model design {
    RefElement insuranceNumber,
    RefElement urgentistId,
    RefElement name,
    RefElement description,
    RefElement Nurse,
    RefElement Staff,
    RefElement leave(),
    RefElement setUp(),
    RefElement interview(),
    RefElement inditePrescription(),
    RefElement payBills(),
    RefElement AdministrativeStaff
},
import model bpm {
    RefElement Triage nurse,
    RefElement Healthcare assistant,
    RefElement Install patient,
    RefElement Perform administrative interview,
    RefElement Perform medical interview,
    RefElement Write prescription,
    RefElement Diagnose,
    RefElement Release patient,
    RefElement Secretary,
    RefElement Retrieve payment for patient invoice
},
import model form {
    RefElement Description,
    RefElement Abbreviation,
    RefElement First Name,
    RefElement Last Name,
    RefElement Urgentist Id,
    RefElement Patient Record Number
}
create M1C define
Correspondence c1 basedOn Generalization Between (Triage nurse) (Nurse) with: weight = 2,
Correspondence c2 basedOn Generalization Between (Healthcare assistant) (Staff) with: weight = 2,
Correspondence c3 mandatory basedOn Induction Between (Release patient) (leave()) with: weight = 4,
Correspondence c4 mandatory basedOn Induction Between (Install patient) (setUp()) with: weight = 4,
Correspondence c5 mandatory basedOn Induction Between (Perform administrative interview) (interview) with: weight = 4,
Correspondence c6 mandatory basedOn Induction Between (Perform medical interview) (interview) with: weight = 4,
Correspondence c7 mandatory basedOn Induction Between (Write prescription) (indite Prescription) with: weight = 4,
Correspondence c8 mandatory basedOn Requirement  Between (Diagnose),(Description and Abbreviation) with: weight = 5,
Correspondence c9 basedOn Aggregation Between (First Name and Last Name) (name) with: weight = 2,
Correspondence c10 basedOn Similarity  Between (Urgentist Id),(urgentistId) with : bidirectional weight = 1,
Correspondence c11 basedOn Similarity  Between (Patient Record Number),(insuranceNumber) with : bidirectional weight = 1,
Correspondence c12 basedOn Similarity  Between (Description),(description) with : bidirectional weight = 1,
Correspondence c13 mandatory basedOn Induction Between (Retrieve payment for patient invoice) (cashCollection()) with: weight = 4,
Correspondence c14 mandatory basedOn Induction Between (Retrieve payment for patient invoice) (payBills()) with: weight = 4,
Correspondence c15 basedOn Generalization Between (Secretary) (AdministrativeStaff) with: weight = 2;
```

Figure VII-20 : Nouvelle version (format textuel) du M1C du SU

VII.5. Conclusion

Dans ce chapitre nous avons présenté le déroulement de notre approche et sa validation conceptuelle ainsi que son implémentation sur un domaine d'application réel (élaboré en collaboration avec le CHU Montpelier) traitant le service d'urgence d'un hôpital. Nous avons tout d'abord présenté les phases mises en place pour la création du M1C, puis les phases appliquées pour le maintien de la cohérence suite aux simulations d'évolution de modèles.

Dans le cas où un nouveau domaine métier est ajouté, le processus de mise en correspondance n'est pas repris en entier. Il est possible d'importer un modèle M2C existant (cf. le processus présenté dans la section IV.4) et d'ajouter uniquement les HLCs impliquant le nouveau métamodèle de domaine, puis de procéder à la phase de raffinement pour créer le modèle M1C. De même, si un nouveau modèle appartenant à un domaine métier existant est ajouté, l'unique travail à faire est de relancer la phase de raffinement étant donné que le M2C reste intact.

Parmi les modèles que nous pourrions ajouter, qui sont conformes à des DSL particuliers, on peut citer par exemple :

- Modèle des directives de santé publiques : pour permettre la représentation des lois et des décrets régissant le fonctionnement de la santé dans un pays donné,
- Modèle de la cartographie des services médicaux : pour permettre la représentation des services médicaux (incluant le personnel affecté) et des équipements disponibles sur place (radiologie, scanner, IRM, analyses biologiques, bloc opératoire, hélicoptère, etc.) ou dans une zone géographique proche (hôpital, clinique, etc.),
- Modèle de facturation : pour inclure les données comptables permettant de facturer les prestations au patient et/ou de traiter avec sa mutuelle le cas échéant, selon la législation en vigueur. Ces données exploitent en particulier les cotations effectuées par les médecins ayant traité le patient considéré.

Le chapitre suivant conclut ce manuscrit en incluant une discussion et une présentation des perspectives de recherche.

CHAPITRE VIII. CONCLUSION ET PERSPECTIVES

"In nature there are no rewards or punishments, there are only consequences"

— *Horace Annesly Vachell*

Dans ce chapitre, nous faisons le bilan des contributions de cette thèse (cf. section VIII.1), nous discutons des limites et des perspectives ouvertes par notre travail (cf. section VIII.2) et nous présentons la liste de nos publications.

VIII.1. Conclusion

Cette thèse s'inscrit dans la continuité des travaux de notre équipe portant sur la multi-modélisation de systèmes complexes. Elle s'intéresse plus précisément à la mise en cohérence de modèles partiels. Contrairement à l'approche VUML (Nassar, 2003) développée antérieurement dans notre équipe, les modèles que nous avons considérés dans cette thèse ne sont pas unifiés par un même langage, mais décrits dans des langages dédiés aux différents domaines d'expertise d'un système complexe. L'utilisation de langages dédiés permet de simplifier la réalisation des modèles d'analyse et de conception, mais rend difficile leur cohabitation pour la compréhension du fonctionnement du système.

Ce travail de recherche a permis de contribuer dans le domaine de la modélisation des systèmes complexes en répondant aux objectifs suivants :

(i) Proposition d'une démarche de mise en correspondance de modèles hétérogènes,

(ii) Proposition d'une démarche de maintien de la cohérence des modèles vis-à-vis du modèle global composé,

(iii) Mise en oeuvre d'un outil support permettant de mettre en place (i) et (ii),

(iv) Application de (i), (ii) et (iii) sur un cas d'étude représentatif.

Le premier défi (i) a été abordé dans le Chapitre IV en proposant un processus de mise en correspondance de modèles hétérogènes produisant un modèle de correspondance. Cette phase s'appuie sur un métamodèle de correspondance que nous avons proposé et sur la définition d'une relation de «Raffinement» entre deux niveaux de modélisation permettant de déduire les

correspondances entre des modèles à partir des correspondances établies entre leurs métamodèles. Le raffinement s'appuie sur une sémantique des types de relation. Doter les types de relation de sémantique est un point important de notre approche puisqu'elle permet bien entendu de donner du sens aux correspondances, mais aussi de servir de filtre lors de la génération des correspondances de niveau modèle. Le résultat global est le renforcement de la sémantique du modèle global produit par notre processus. Nous donnons la possibilité à l'expert d'associer une expression sémantique à chaque type de relation par le biais du DSL SED (cf. section IV.7.3.2). Ce DSL offre la possibilité de décrire le corps de la condition dans un langage donné (Java, OCL, etc.) en manipulant directement les éléments du modèle ou bien des ontologies (pour lesquelles il est nécessaire de réaliser une transformation du domaine des modèles vers le domaine des ontologies). Les conditions ne provoquent pas d'effet de bord et permettent de distinguer, parmi les correspondances produites de façon combinatoires lors de l'opération de reproduction, les correspondances fausses des correspondances valides. Le résultat de cette phase est un modèle global virtuel.

Le second défi (ii) a été abordé dans le Chapitre V. Il consiste à exploiter le modèle de correspondance précédent afin de traiter les changements identifiés de façon automatique sur les modèles partiels de façon à maintenir la cohérence des modèles interconnectés. Une fois les changements identifiés, le processus de maintien de la cohérence procède à leur classification, puis les différents impacts potentiels sont identifiés automatiquement ainsi que la présence éventuelle de cycles. Ces derniers sont gérés à l'aide de l'expert. L'ordre de traitement des évolutions des modèles est important car sans coordination, le traitement des évolutions pourrait devenir ingérable. Pour cela, selon la stratégie choisie, une liste de changements est générée et ordonnée selon un calcul de coefficients de pondération. Enfin, les changements sont traités de façon automatique en s'appuyant sur les sémantiques attribuées aux types de relation.

Le troisième défi (iii), décrit dans le Chapitre VI, a consisté à développer un outil support sous l'environnement Eclipse appelé HMCS (*Heterogeneous Matching and Consistency Management Suite*). En partant d'un ensemble de modèles et de leurs métamodèles respectifs, HMCS permet d'assister l'expert intégrateur dans la création d'un modèle global virtuel et dans le maintien de sa cohérence suite aux éventuelles évolutions des modèles partielles.

Le quatrième défi (iv), a été décrit dans le Chapitre VII. Il a consisté à traiter une étude de cas représentative des problèmes que nous cherchons à résoudre et d'y appliquer, par l'intermédiaire de l'outil HMCS, les propositions présentées dans (i) et (ii). Le cas d'étude choisi est celui d'un service d'urgence d'un hôpital, élaboré en collaboration avec le CHU de Montpellier.

VIII.2. Limites de notre approche et perspectives

Notre approche ne prétend pas répondre de façon parfaite et complétement opérationnelle à la problématique traitée. Certaines limitations existent tant sur le plan théorique que sur le plan de l'outillage.

D'un point de vue théorique, une première limitation s'inscrit au niveau du processus de gestion de la cohérence (cf. section V.7). Dans ce processus nous nous sommes basés sur une formule simplifiée de calcul des coefficients de pondération. Pour améliorer cette formule, nous sommes en train de mettre en place la théorie de marche aléatoire (*random walk*) (Fouss et al.,

2007) afin d'évaluer la probabilité qu'un élément dans le modèle de correspondance M1C soit impacté par un changement. Par ailleurs, les correspondances que nous avons traitées mettent en œuvre des éléments de type EClass ou de type EAttribute. Nous n'avons pas envisagé d'exprimer des correspondances sur des éléments de type EReference, étant donné que les domaines étudiés ne nécéssitent pas ce type de correspondance. Nous pensons que le processus proposé n'en serait pas affecté, mais nous envisageons néanmoins d'étudier ce cas. Une deuxième limitation réside dans le fait que nous sommes partis du principe que les tâches non automatisables du processus sont effectuées par un expert ayant une connaissance globale des différents modèles partiels, et donc capable d'identifier avec précision les correspondances. Cette hypothèse rend le processus très dépendant de cet expert et donc relativement centralisé. Dans la réalité des développements industriels, ce travail est réalisé par plusieurs concepteurs qui travaillent de façon collaborative. Nous avons ainsi initié une étude pour transformer le processus actuel en un processus collaboratif. Cette transformation se traduira entre autres par une redéfinition des tâches à effectuer par les différents concepteurs, et par l'utilisation d'outils collaboratifs.

D'un point de vue outillage, le prototype HMCS est opérationnel, mais des améliorations sont néanmoins à envisager au niveau de l'implémentation. Premièrement, les facteurs *taille* et *temps de création* du modèle de correspondance par le module RT risquent de ne pas être négligeables lors du passage à l'échelle. Concernant le premier facteur, la taille du modèle de correspondance sera immédiatement réduite par les modules MFT et OFT, où les sémantiques des différents types de relation sont évaluées, ce qui permettra d'enlever les correspondances non valides. Pour le second facteur, nous travaillons à intégrer le module ReproT dans le module MFT (et OFT) de telle sorte qu'une correspondance ne soit créée que si elle répond aux exigences du module MFT (ou OFT) en termes de sémantiques de types de relation. À partir de ce moment là, il sera possible de compléter l'évaluation de la section IV.8 en incluant les types de relation non traités. Un autre aspect de l'évaluation est d'étudier dans quels cas de systèmes complexes l'approche manuelle pourrait être considérée plus efficace que l'approche de mise en correspondance semi-automatique proposée. Deuxièmement, les types de relation ajoutés au MMC sont greffés automatiquement dans la grammaire Xtext de l'éditeur textuel du Framework AMT. Cela nous contraint à le recompiler pour que ces types de relations puissent être utilisés. Une piste possible serait de modifier, à l'exécution, l'arbre syntaxique abstrait (AST : *Abstract Syntax Tree*). Troisièmement, l'accès au modèle de correspondance est verrouillé dès le moment où il est en modification (réception des notifications par exemple) et il ne peut plus être sujet d'autres modifications. De ce fait, il faut attendre que le modèle soit de nouveau disponible avant de faire d'autres modifications, sous peine de risquer de les perdre. Pour assurer l'accès simultané, nous proposons d'utiliser EMFcollab qui permet de garantir une synchronisation collaborative en temps réel. Quatrièmement, il faut améliorer l'ergonomie de l'outil afin de pouvoir le mettre à disposition de la communauté Eclipse. Cinquièmement, nous sommes en train de développer un plug-in permettant de générer partiellement le code applicatif du système en Java à partir du modèle de correspondance.

Enfin pour conclure ce manuscrit, nous sommes conscients que l'étude de cas choisie pour la validation de notre approche (cf. Chapitre VII) possède des limites d'où la nécessité d'une étude empirique plus poussée concernant le passage à l'échelle. Nous cherchons à cet effet un domaine d'application industriel pour appliquer et renforcer la validation de notre approche.

VIII.3. Liste des publications liées à la thèse

Cette section présente par catégorie, les publications que nous avons réalisées pendant cette thèse :

- Revue internationale :
  - El Hamlaoui Mahmoud, Ebersold Sophie, Anwar Adil, Coulette Bernard and Nassar Mahmoud. Towards a framework for heterogeneous models matching. Journal of Software Engineering, Science Alert, New York - USA, (support électronique), 2014.
- Revue nationale :
  - El Hamlaoui Mahmoud, Ebersold Sophie, Coulette Bernard, Anwar Adil, Nassar Mahmoud. Maintien de la cohérence de modèles hétérogènes. Technique et Science Informatiques, Hermès Science, 2015 (à paraître).
- Conférences internationales :
  - El Hamlaoui Mahmoud, Trojahn Cassia, Ebersold Sophie, Coulette Bernard. Towards an Ontology-based Approach for Heterogeneous Model Matching. International Workshop On the Globalization of Modeling Languages (GEMOC 2014, associé à MODELS, Valencia-Spain, 28/09/2014-28/09/2014, CEUR Workshop Proceedings,
  - El Hamlaoui Mahmoud, Ebersold Sophie, Anwar Adil, Coulette Bernard, Nassar Mahmoud. Heterogeneous models matching for consistency management. IEEE International Conference on Research Challenges in Information Science (RCIS 2014) Marrakech-Maroc, 28/06/2014-30/06/2014, IEEE Explore digital library,
  - El Hamlaoui Mahmoud, Ebersold Sophie, Coulette Bernard, Anwar Adil, Nassar Mahmoud. A process for defining a unique correspondence model to relate heterogeneous models. International Conference on Evaluation of Novel Approaches to Software Engineering (ENASE 2013), Angers-France, 04/06/2013-06/06/2013, SciTePress,
  - El Hamlaoui Mahmoud, Ebersold Sophie, Anwar Adil, Nassar Mahmoud, Coulette Bernard. A Process for Maintaining Heterogeneous Models Consistency through Change Synchronization. ACS/IEEE International Conference on Computer Systems and Applications (AICCSA 2013), Ifrane-Morocco, 27/05/2013-30/05/2013, IEEExplore digital library.
- Conférences nationales :
  - El Hamlaoui Mahmoud, Ebersold Sophie, Coulette Bernard, Nassar Mahmoud. Mise en correspondance de modèles hétérogènes par points de vue. Conférence en Ingénierie du Logiciel (CIEL 2012), Rennes, 19/06/12-21/06/12, IRISA,
  - El Hamlaoui Mahmoud, Ebersold Sophie, Anwar Adil, Dkaki Taoufiq, Coulette Bernard, Nassar Mahmoud. A proposition to solve heterogeneous model matching problems. Papier long présenté à CIEL 2015, Bordeaux, 9 juin 2015.

BIBLIOGRAPHIE

Acher, M., Collet, P., Lahire, P. & France, R. 2010. Comparing approaches to implement feature model composition. *Modelling Foundations and Applications*. Springer.

Agilebirds. 2011. *Openflexo* [Online]. Available: http://www.openflexo.com [Accessed June 2015].

Aizenbud-Reshef, N., Nolan, B. T., Rubin, J. & Shaham-Gafni, Y. 2006. Model traceability. *IBM Systems Journal,* 45**,** 515-526.

Alanen, M. & Porres, I. 2003. *Difference and union of models*, Springer.

Albert, M., Muñoz, J., Pelechano, V. & Pastor, Ó. 2006. Model to text transformation in practice: generating code from rich associations specifications. *Advances in Conceptual Modeling-Theory and Practice*. Springer.

Anwar, A. 2009. *Formalisation par une approche IDM de la composition de modèles dans le profil VUML*. Thèse de doctorat, Université de Toulouse II - Le Mirail.

Anwar, A., Ebersold, S., Coulette, B., Nassar, M. & Kriouile, A. 2010. A rule-driven approach for composing viewpoint-oriented models. *Journal of Object Technology,* 9**,** 89-114.

Baker, P., Loh, S. & Weil, F. 2005. Model-Driven engineering in a large industrial context—motorola case study. *Model Driven Engineering Languages and Systems*. Springer.

Banek, M., Vrdoljak, B., Tjoa, A. M. & Skočir, Z. 2007. Automating the schema matching process for heterogeneous data warehouses. *Data Warehousing and Knowledge Discovery*. Springer.

Banerjee, S. & Pedersen, T. 2002. An adapted Lesk algorithm for word sense disambiguation using WordNet. *Computational linguistics and intelligent text processing*. Springer.

Barbier, F., Deltombe, G., Parisy, O. & Youbi, K. Model driven reverse engineering : Increasing legacy technology independence. Second India Workshop on Reverse Engineering, 2011 Thiruvanantpuram. 126-139.

Baudry, B., Combemale, B., Deantoni, J. & Mallet, F. 2012. Action Spécifique 2011 GeMoC du GDR GPL «Ingénierie du logiciel pour les systèmes hétérogènes».

Baudry, B., Ghosh, S., Fleurey, F., France, R., Le Traon, Y. & Mottu, J.-M. 2010. Barriers to systematic model transformation testing. *Communications of the ACM,* 53**,** 139-143.

Bernoussi, I. 2008. Les DSL, un standard dans 2 ans ? *Programmez, le magazine des développeurs*.

Bettini, L. 2013. *Implementing Domain-Specific Languages with Xtext and Xtend*, Packt Publishing Ltd.

Beugnard, A., Dagnat, F., Guérin, S. & Guychard, C. Des situations de modélisation pour évaluer les outils de modélisation. INFORSID 2014: 32ème congrès de l'INFormatique des ORganisations et Systèmes d'Information et de Décision, 2014. 181-196.

Bézivin, J. 2004. In search of a basic principle for model driven engineering. *Novatica Journal, Special Issue,* 5**,** 21-24.

Bézivin, J. 2005. On the unification power of models. *Software & Systems Modeling,* 4**,** 171-188.

Bézivin, J. 2006. Model driven engineering: An emerging technical space. *Generative and transformational techniques in software engineering*. Springer.

Bézivin, J. 2009. Advances in Model Driven Engineering. *Jornadas de Ingeniería del Software y Bases de Datos***,** 8-11.

Bézivin, J., Bouzitouna, S., Del Fabro, M. D., Gervais, M.-P., Jouault, F., Kolovos, D., Kurtev, I. & Paige, R. F. A canonical scheme for model composition. Model Driven Architecture–Foundations and Applications, 2006. Springer, 346-360.

Bézivin, J. & Gerbé, O. Towards a precise definition of the OMG/MDA Framework. Automated Software Engineering (ASE), 2001 San Diego (USA). IEEE, 273-280.
Bhave, A., Krogh, B. H., Garlan, D. & Schmerl, B. View consistency in architectures for cyber-physical systems. International Conference on Cyber-Physical Systems (ICCPS), 2011. IEEE/ACM, 151-160.
Bluage. 2015a. *Blu Age Forward* [Online]. Available: http://www.bluage.com/en/en_product/en-ba-forward.html [Accessed April 2015].
Bluage. 2015b. *Blu Age Reverse Modeling* [Online]. Available: http://www.nomagic.com/products/magicdraw-addons/mdd-integrations/blu-age/reverse.html [Accessed April 2015].
Bohlen, M. 2007. *AndroMDA* [Online]. Available: http://www.andromda.org [Accessed February 2014].
Boronat, A., Carsí, J. Á., Ramos, I. & Letelier, P. 2007. Formal model merging applied to class diagram integration. *Electronic Notes in Theoretical Computer Science,* 166**,** 5-26.
Boronat, A., Knapp, A., Meseguer, J. & Wirsing, M. 2009. What is a multi-modeling language? *Recent Trends in Algebraic Development Techniques.* Springer.
Boulanger, F., Jacquet, C., Hardebolle, C. & Rouis, E. 2010. Modeling heterogeneous points of view with ModHel'X. *Models in Software Engineering.* Springer.
Bousetta, B., El Beggar, O. & Gadi, T. 2013. A methodology for CIM modelling and its transformation to PIM. *Journal of Information Engineering and Applications,* 3**,** 1-21.
Brun, C. & Pierantonio, A. 2008. Model differences in the eclipse modeling framework. *UPGRADE, The European Journal for the Informatics Professional,* 9**,** 29-34.
Bruneliere, H., Cabot, J., Clasen, C., Jouault, F. & Bézivin, J. 2010. Towards model driven tool interoperability: bridging eclipse and microsoft modeling tools. *Modelling foundations and applications.* Springer.
Buneman, P., Davidson, S. & Kosky, A. Theoretical aspects of schema merging. Extending DataBase Technology (EDBT), 1992. Springer, 152-167.
Burden, H., Heldal, R. & Whittle, J. Comparing and contrasting model-driven engineering at three large companies. International Symposium on Empirical Software Engineering and Measurement (ESEM), 2014 Torino (Italy). ACM.
Burns, E., Schalk, C. & Griffin, N. 2009. *JavaServer Faces 2.0*, McGraw-Hill.
Cabot, J. 2015. Clarifying concepts: MBE vs MDE vs MDD vs MDA. Available from: http://modeling-languages.com/clarifying-concepts-mbe-vs-mde-vs-mdd-vs-mda/ [Accessed April 2015].
Cariou, E., Belloir, N. & Barbier, F. Contrats de transformation pour la validation de raffinement de modèles. 5ème journée sur l'Ingénierie Dirigée par les Modèles (IDM), 2009 Nancy, France. 1-16.
Castano, S., De Antonellis, V. & De Capitani Di Vimercati, S. 2001. Global Viewing of Heterogeneous Data Sources. *IEEE Trans. on Knowl. and Data Eng.,* 13**,** 277-297.
Cavallaro, L., Di Nitto, E., Furia, C. A. & Pradella, M. A Tile-Based Approach for Self-Assembling Service Compositions. International Conference on Engineering
of Complex Computer Systems (ICECCS), 2010. IEEE, 43-52.
Cdo, E. 2014. *CDO Model Repository Overview* [Online]. Available: http://www.eclipse.org/cdo/ [Accessed October
2014].
Charlet, J., Bachimont, B. & Troncy, R. 2004. Ontologies pour le Web sémantique. *Revue Information, Interaction, Intelligence I3*.
Cheatham, M. & Hitzler, P. 2013. The role of string similarity metrics in ontology alignment. Technical report, Kno.e.sis Center, Wright State University.
Cicchetti, A. & Ciccozzi, F. Towards a Novel Model Versioning Approach Based on the Separation Between Linguistic and Ontological Aspects. Models and Evolution (ME@MoDELS), 2013. IEEE, 60-69.
Cicchetti, A., Di Ruscio, D., Eramo, R. & Pierantonio, A. Automating co-evolution in model-driven engineering. Enterprise Distributed Object Computing (EDOC) Conference 2008. IEEE, 222-231.
Cicchetti, A., Di Ruscio, D. & Pierantonio, A. 2007. A Metamodel Independent Approach to Difference Representation. *Journal of Object Technology,* 6**,** 165-185.

Clasen, C., Jouault, F. & Cabot, J. Virtual Composition of EMF Models. 7ème Journée sur l'Ingénierie Dirigée par les Modèles (IDM), 2011a Lille (France).
Clasen, C., Jouault, F. & Cabot, J. 2011b. VirtualEMF: a model virtualization tool. *Advances in Conceptual Modeling. Recent Developments and New Directions.* Springer.
Conradi, R. & Westfechtel, B. 1998. Version models for software configuration management. *ACM Computing SURveys (CSUR),* 30**,** 232-282.
Cruz, I. F., Palandri Antonelli, F. & Stroe, C. Efficient selection of mappings and automatic quality-driven combination of matching methods. International Workshop on Ontology Matching (IWOM@ISWC) CEUR Workshop Proceedings, 2009. Citeseer, 49-60.
Cruz, I. F. & Sunna, W. 2008. Structural alignment methods with applications to geospatial ontologies. *Transactions in Geographic Information Science (GIS),* 12**,** 683-711.
Del Fabro, M. D., Bezivin, J., Jouault, F., Breton, E. & Gueltas, G. AMW: a generic model weaver. 1ère Journée sur l'Ingénierie Dirigée par les Modèles (IDM 2005), 2005. 7-11.
Del Fabro, M. D. & Valduriez, P. Semi-automatic model integration using matching transformations and weaving models. ACM Symposium on Applied Computing (SAC), 2007 Seoul (Korea). ACM, 963-970.
Del Fabro, M. D. & Valduriez, P. 2009. Towards the efficient development of model transformations using model weaving and matching transformations. *Software & Systems Modeling,* 8**,** 305-324.
Delligatti, L. 2013. *SysML Distilled: A Brief Guide to the Systems Modeling Language*, Addison-Wesley.
Descartes, R. & Gröber, G. 1905. *Discours de la méthode: 1637*, Heitz.
Di Ruscio, D., Iovino, L. & Pierantonio, A. What is needed for managing co-evolution in MDE? nternational Workshop on Model Comparison in Practice (IWMCP), 2011. ACM, 30-38.
Dingel, J., Diskin, Z. & Zito, A. 2008. Understanding and improving UML package merge. *Software & Systems Modeling,* 7**,** 443-467.
Do, H.-H., Melnik, S. & Rahm, E. 2003. Comparison of schema matching evaluations. *Web, Web-Services, and Database Systems.* Springer.
Do, H. H. 2007. *Schema Matching and Mapping-based Data Integration: Architecture, Approaches and Evaluation*, VDM Verlag.
Dreßler, K. & Ngomo, A.-C. N. 2014. On the Efficient Execution of Bounded Jaro-Winkler Distances.
Drey, Z., Faucher, C., Fleurey, F., Mahé, V. & Vojtisek, D. 2009a. Kermeta language. *Reference Manual.*
Drey, Z., Faucher, C., Fleurey, F., Mahé, V. & Vojtisek, D. 2009b. *Kermeta language - Reference manual* [Online]. Available: http://www.kermeta.org/documents/user_doc/manual/ [Accessed January 2015].
Dupuy-Chessa, S., Combemale, B., Gervais, M.-P., Nodenot, T., Le Pallec, X. & Wouters, L. Vers une approche centrée humain pour la définition de langages de modélisation graphiques. 32ème congrès Inforsid, 2014. 79-94.
Eclipse. 2011a. *Eclipse4/RCP/Architectural Overview* [Online]. Available: https://wiki.eclipse.org/Eclipse4/RCP/Architectural_Overview [Accessed November 2014].
Eclipse. 2011b. *Java Emitter Templates* [Online]. Available: http://www.eclipse.org/modeling/m2t/?project=jet [Accessed 2014 November].
Egyed, A. & Medvidovic, N. A formal approach to heterogeneous software modeling. Fundamental Approaches to Software Engineering (FASE), 2000 Berlin (Germany). Springer, 178-192.
Eker, J., Janneck, J. W., Lee, E. A., Liu, J., Liu, X., Ludvig, J., Neuendorffer, S., Sachs, S. & Xiong, Y. 2003. Taming heterogeneity-the Ptolemy approach. *Proceedings of the IEEE,* 91**,** 127-144.
Evans, E. 2004. *Domain-driven design: tackling complexity in the heart of software*, Addison-Wesley Professional.
Fabro, M. D. D. 2008. *AMW Use Case - Tool Interoperability of bug tracking tools* [Online]. Available: http://www.eclipse.org/gmt/amw/usecases/interoperability/ [Accessed December 2013].
Falleri, J.-R., Huchard, M., Lafourcade, M. & Nebut, C. 2008. Metamodel matching for automatic model transformation generation. *Model Driven Engineering Languages and Systems.* Springer.
Farail, P., Goutillet, P., Canals, A., Le Camus, C., Sciamma, D., Michel, P., Crégut, X. & Pantel, M. 2006. The TOPCASED project: a toolkit in open source for critical aeronautic systems design. *Ingenieurs de l'Automobile***,** 54-59.
Favre, J.-M. Meta-model and model co-evolution within the 3D software space. Workshop on Evolution of Large-scale Industrial Software Applications (ELISA@ICSM), 2003 Amsterdam (Netherlands). 98-109.

Fenza, G., Loia, V. & Senatore, S. 2008. A hybrid approach to semantic web services matchmaking. *International Journal of Approximate Reasoning,* 48**,** 808-828.
Filman, R., Elrad, T. & Clarke, S. 2004. *Aspect-oriented software development*, Addison-Wesley Professional.
Fleurey, F., Baudry, B., France, R. & Ghosh, S. 2008. A generic approach for automatic model composition. *Models in software engineering*. Springer.
Fouss, F., Pirotte, A., Renders, J.-M. & Saerens, M. 2007. Random-walk computation of similarities between nodes of a graph with application to collaborative recommendation. *IEEE Transactions on Knowledge and Data Engineering (TKDE),* 19**,** 355-369.
Fowler, M. 2010. *Domain-Specific Languages* Addison-Wesley Professional.
Friedenthal, S., Griego, R. & Sampson, M. INCOSE Model Based Systems Engineering (MBSE) initiative. International Council on Systems Engineering (INCOSE) Symposium, 2007 San Diego (USA).
Gabriel, R. P., Northrop, L., Schmidt, D. C. & Sullivan, K. Ultra-large-scale systems. Object-Oriented Programming Systems, Languages, and Applications (OOPSLA@SIGPLAN), 2006 New York, (USA). ACM, 632-634.
Gamma, E., Helm, R., Johnson, R. & Vlissides, J. 1994. *Design patterns: elements of reusable object-oriented software*, Pearson Education.
Garcés, K., Jouault, F., Cointe, P. & Bézivin, J. A domain specific language for expressing model matching. 5ème Journée sur l'Ingénierie Dirigée par les Modèles (IDM), 2009 Nancy, France. 33-48.
Garces, K., Kling, W. & Jouault, F. Automatizing the evaluation of model matching systems. Workshop on Matching and Meaning (WMM@AISB), 2010 Leicester (UK). To appear.
Gemoc. 2012. *Initiative On the Globalization of Modeling Languages* [Online]. Available: http://gemoc.org/ins/ [Accessed March 2015].
Google. 2010. *CodePro Analytix for Eclipse, Rational and Websphere* [Online]. Available: https://google-web-toolkit.googlecode.com/files/CodePro-EvalGuide.pdf [Accessed November 2014].
Gpl, G. *Groupement de Recherche Génie de la Programmation et du Logiciel* [Online]. Available: http://gdr-gpl.cnrs.fr/ [Accessed March 2015].
Gray, J., Lin, Y. & Zhang, J. 2006. Automating change evolution in model-driven engineering. *Computer,* 39**,** 51-58.
Greenfields-Development. 2015. *MyAppConverter* [Online]. Available: https://www.myappconverter.com/ [Accessed April 2015].
Gronback, R. C. 2009. *Eclipse modeling project: a domain-specific language (DSL) toolkit*, Pearson Education.
Gruber, T. R. 1995. Toward principles for the design of ontologies used for knowledge sharing? *International journal of human-computer studies,* 43**,** 907-928.
Gruschko, B., Kolovos, D. & Paige, R. Towards synchronizing models with evolving metamodels. International Workshop on Model-Driven Software Evolution (MDSE@CSMR), 2007.
Guerin, S. 2013. *Tutorial 4: Using the "CityMapping" demonstrator (pure EMF context)* [Online]. Available: https://openflexo.org/tiki/article4 [Accessed April 2015].
Guychard, C., Guerin, S., Koudri, A., Beugnard, A. & Dagnat, F. Conceptual interoperability through Models Federation. Semantic Information Modeling for Federation (SIMF@MODELS), 2013 Miami (USA). CEUR, .
Hailpern, B. & Tarr, P. 2006. Model-driven development: The good, the bad, and the ugly. *IBM systems journal,* 45**,** 451-461.
Hausmann, J. H. & Kent, S. Visualizing model mappings in UML. Software Visualization (SoftVis), 2003 San Diego (USA). ACM, 169-178.
Herrmann, C., Krahn, H., Rumpe, B., Schindler, M. & Völkel, S. An algebraic view on the semantics of model composition. European Conference on Model Driven Architecture-Foundations and Applications (ECMDA-FA), 2007 Haifa (Israel). Springer, 99-113.
Herrmannsdoerfer, M. Solving the ttc 2011 reengineering case with edapt. Transformation Tool Contest (TTC), 2011. 149-158.
Herrmannsdoerfer, M., Benz, S. & Juergens, E. COPE: A language for the coupled evolution of metamodels and models. Workshop on Model Co-Evolution and Consistency Management (MCCM@MODELS), 2008 Toulouse (France).
Hilliard, R. Viewpoint modeling. Workshop on Describing Software Architecture with UML (DSAU@ICSE), 2001 Toronto (Canada). ACM.

Hirst, G. & St-Onge, D. 1998. Lexical chains as representations of context for the detection and correction of malapropisms. *WordNet: An electronic lexical database,* 13**,** 305-332.
Hutchinson, J., Whittle, J. & Rouncefield, M. 2014. Model-driven engineering practices in industry: Social, organizational and managerial factors that lead to success or failure. *Science of Computer Programming,* 89**,** 144-161.
Iwu, F. 2014. *KOMMA* [Online]. Available: http://komma.enilink.net/ [Accessed November 2014].
Jenkinson, T., Truss, J. & Seidel, D. 2012. Countable homogeneous multipartite graphs. *European Journal of Combinatorics,* 33**,** 82-109.
Jézéquel, J.-M., Combemale, B. & Vojtisek, D. 2012. *Ingénierie Dirigée par les Modèles: des concepts à la pratique*, Ellipses.
Jiang, Y. 1998. *Using Heuristic Approaches to Detect Record Boundaries in Semistructured Web Documents.* Citeseer.
Jouault, F., Allilaire, F., Bézivin, J. & Kurtev, I. 2008. ATL: A model transformation tool. *Science of computer programming,* 72**,** 31-39.
Kalfoglou, Y. & Schorlemmer, M. 2002. Information-flow-based ontology mapping. *On the move to meaningful internet systems 2002: CoopIS, DOA, and ODBASE*. Springer.
Kappel, G., Kargl, H., Kramler, G., Schauerhuber, A., Seidl, M., Strommer, M. & Wimmer, M. Matching Metamodels with Semantic Systems-An Experience Report. BTW Workshops, 2007. 38-52.
Kardoš, M. & Drozdová, M. 2010. Analytical method of CIM to PIM transformation in Model Driven Architecture (MDA). *Journal of Information and Organizational Sciences,* 34**,** 89-99.
Kent, S. Model driven engineering. Integrated Formal Methods (IFM), 2002 Turku (Finland) Springer, 286-298.
Kherraf, S., Lefebvre, É. & Suryn, W. Transformation from CIM to PIM using patterns and archetypes. Australian SoftWare Engineering Conference (ASWEC), 2008. IEEE, 338-346.
Kiczales, G., Lamping, J., Mendhekar, A., Maeda, C., Lopes, C., Loingtier, J.-M. & Irwin, J. 1997. *Aspect-oriented programming*, Springer.
Klatt, B. 2007. Xpand: A closer look at the model2text transformation language. *Language* [Online], 10. Available: http://bar54.de/benjamin.klatt-Xpand.pdf
Kleppe, A. G., Warmer, J. B. & Bast, W. 2003. *MDA explained: the model driven architecture: practice and promise*, Addison-Wesley Professional.
Koegel, M., Herrmannsdoerfer, M., Helming, J. & Li, Y. State-based vs. operation-based change tracking. Workshop on Models and Evolution (WMM@MODELS), 2009 Denver (USA).
Kolovos, D., Rose, L., Paige, R. & Garcıa-Domınguez, A. 2010. The epsilon book. *Structure,* 178**,** 1-10.
Kolovos, D. S., Paige, R. F. & Polack, F. A. 2006a. Merging models with the epsilon merging language (EML). *Model driven engineering languages and systems.* Springer.
Kolovos, D. S., Paige, R. F. & Polack, F. A. Model comparison: a foundation for model composition and model transformation testing. Workshop on Global Integrated Model Management (GaMMa@ICSE), 2006b Shanghai (China). ACM, 13-20.
Königs, A. & Schürr, A. 2006. Mdi: A rule-based multi-document and tool integration approach. *Software & Systems Modeling,* 5**,** 349-368.
Koning, H. & Van Vliet, H. 2006. A method for defining IEEE Std 1471 viewpoints. *Journal of Systems and Software,* 79**,** 120-131.
Koster, V. 2007. Implementation and integration of a domain specific language with oaw and xtext. *MT AG, Ratingen.*
Lakhrissi, Y. 2010. *Intégration de la modélisation comportementale dans la conception par points de vue.* Thèse de doctorat, Université Toulouse II - Le Mirail.
Larman, C. 2005. *Applying UML and patterns: an introduction to object-oriented analysis and design and iterative development, 3/e*, Pearson Education India.
Leacock, C. & Chodorow, M. 1998. Combining local context and WordNet similarity for word sense identification. *WordNet: An electronic lexical database,* 49**,** 265-283.
Letkeman, K. 2015. *Comparing and merging UML models in IBM Rational Software Architect* [Online]. Available: http://www.ibm.com/developerworks/rational/tutorials/realign-your-models-after-migration-or-transformation/ [Accessed February 2015].
Lin, D. An information-theoretic definition of similarity. International Conference on Maching Learning (ICML), 1998 Wisconsin (USA). 296-304.
Lin, Y., Gray, J. & Jouault, F. 2007. DSMDiff: a differentiation tool for domain-specific models. *European Journal of Information Systems,* 16**,** 349-361.

Liu, N., Grundy, J. & Hosking, J. A visual language and environment for composing web services. International Conference on Automated Software Engineering (ASE), 2005 California (USA). ACM, 321-324.
Lopes, D., Hammoudi, S. & Abdelouahab, Z. 2006a. Schema matching in the context of model driven engineering: From theory to practice. *Advances in Systems, Computing Sciences and Software Engineering.* Springer.
Lopes, D., Hammoudi, S., Bézivin, J. & Jouault, F. 2006b. Mapping specification in MDA: From theory to practice. *Interoperability of Enterprise Software and Applications.* Springer.
Malone, M. W. 2014. *Process subversion in Agile Scrum software development: A phenomenological approach.* Capella University.
Medvidovic, N. & Taylor, R. N. 2000. A classification and comparison framework for software architecture description languages. *IEEE Transactions on Software Engineering (TSE),* 26**,** 70-93.
Mellor, S. J., Clark, T. & Futagami, T. 2003. Model-driven development: guest editors' introduction. *IEEE software,* 20**,** 14-18.
Melnik, S., Garcia-Molina, H. & Rahm, E. Similarity flooding: A versatile graph matching algorithm and its application to schema matching. International Conference on Data Engineering (ICDE), 2002 California (USA). IEEE, 117-128.
Mens, T. & Van Gorp, P. 2006. A taxonomy of model transformation. *Electronic Notes in Theoretical Computer Science,* 152**,** 125-142.
Minsky, M. 1965. Matter, mind and models. *International Federation for Information Processing (IFIP),* 1**,** 45-50.
Modica, T., Biermann, E. & Ermel, C. 2009. An Eclipse Framework for Rapid Development of Rich-featured GEF Editors based on EMF Models. *GI Jahrestagung,* 154**,** 2972-2985.
Moody, D. L. 2009. The "physics" of notations: toward a scientific basis for constructing visual notations in software engineering. *IEEE Transactions on Software Engineering (TSE),* 35**,** 756-779.
Moreira, T. G., Wehrmeister, M., Pereira, C. E., Pétin, J.-F. & Levrat, E. Automatic code generation for embedded systems: From UML specifications to VHDL code. International Conference on Industrial Informatics (INDIN), 2010 Osaka (Japan). IEEE, 1085-1090.
Moriconi, M., Qian, X. & Riemenschneider, R. A. 1995. Correct architecture refinement. *IEEE Transactions on Software Engineering (TSE),* 21**,** 356-372.
Morin, B., Klein, J., Barais, O. & Jézéquel, J.-M. A generic weaver for supporting product lines. International workshop on Early Aspects (EA@ICSE), 2008 Leipzig (Germany). ACM, 11-18.
Mosser, S., Bergel, A. & Blay–Fornarino, M. Visualizing and assessing a compositional approach of business process design. International Conference on Software Composition (SC), 2010 Malaga (Spain). Springer, 90-105.
Musset, J., Juliot, É., Lacrampe, S., Piers, W., Brun, C., Goubet, L., Lussaud, Y. & Allilaire, F. 2006. Acceleo user guide. Available: http://www.acceleo.org/doc/obeo/en/acceleo-2.6-user-guide.pdf.
Nassar, M. VUML: a Viewpoint oriented UML Extension. International Conference on Automated Software Engineering (ASE), 2003 Montreal (Canada). IEEE, 373-376.
Nassar, M. 2005. *Analyse/conception par points de vue: le profil VUML.* Thèse de doctorat, Université Toulouse II - Le Mirail.
Noy, N. F. & Musen, M. A. Algorithm and tool for automated ontology merging and alignment. AAAI Conference on Artificial Intelligence, 2000 Austin (Texas).
Ohst, D., Welle, M. & Kelter, U. 2003. Differences between versions of UML diagrams. *ACM SIGSOFT Software Engineering Notes,* 28**,** 227-236.
Oliveira, K. S. & De Oliveira, T. C. A guidance for model composition. International Conference on Software Engineering Advances (ICSEA), 2007 Cap Esterel (France). IEEE, 27-27.
OMG. 1989. *Object Management Group* [Online]. Available: http://www.omg.org [Accessed April 2014].
OMG. 2011a. *Abstract Syntax Tree Metmodel (ASTM)* [Online]. Available: http://www.omg.org/spec/ASTM/1.0.
OMG. 2011b. *Knowledge Discovery Metamodel (KDM)* [Online]. Available: http://www.omg.org/spec/KDM/1.3 [Accessed March 2014].
OMG. 2011c. *Unified Modeling Language (UML), version 2.4.1* [Online]. Available: http://www.omg.org/spec/UML/2.4.1/Superstructure/PDF [Accessed November 2014].
OMG. 2013. *Business Process Model And Notation (BPMN), version 2.0.2* [Online]. Available: http://www.omg.org/spec/BPMN/2.0.2/PDF [Accessed February 2014].

OMG. 2014. *Metadata Interchange (XMI), version 2.4.2* [Online]. Available: http://www.istr.unican.es/pyemofuc/PyEmofUCFiles/XMI_formal-14-04-04.pdf [Accessed June 2014].
OMG. 2015. *Meta Object Facility (MOF), Query/View/Transformation (QVT), version 2.0* [Online]. Available: http://www.omg.org/spec/QVT/1.2 [Accessed March 2015].
Papagelis, M., Plexousakis, D. & Nikolaou, P. N. 2005. CONFIOUS: Managing the electronic submission and reviewing process of scientific conferences. *Web Information Systems Engineering–WISE 2005*. Springer.
Parreiras, F. S., Staab, S. & Winter, A. 2007. *TwoUse: Integrating UML models and OWL ontologies*, Inst. für Informatik.
Patil, L. & Atique, M. A novel feature selection based on information gain using WordNet. Science and Information Conference (SAI), 2013, 2013 London (UK). IEEE, 625-629.
Pérez-Martínez, J. E. & Sierra-Alonso, A. From analysis model to software architecture: A PIM2PIM mapping. Model Driven Architecture – Foundations and Applications (ECMDA-FA), 2006 Biblao (Spain). Springer, 25-39.
Perin, M. & Wouters, L. Using Ontologies for Solving Cross-Domain Collaboration Issues. World Congress, 2014. 7837-7842.
Pillar, I. 2007. *Integrating Continuity of Operations (COOP) into the Enterprise Architecture* [Online]. Available: http://www.juniper.net/us/en/local/pdf/resource-guides/9050014-en.pdf [Accessed June 2014].
Qgears_Kft. 2010. *EMFCollab: collaborative editing for emf models* [Online]. Available: http://qgears.com/products/emfcollab/, [Accessed November 2014].
Rahm, E. & Bernstein, P. A. 2001. A survey of approaches to automatic schema matching. *the VLDB Journal,* 10**,** 334-350.
Rausch, A., Rumpe, B. & Hoogendoorn, L. Aspect-oriented framework modeling. Workshop on Aspect-Oriented Software Development (AOSD@UML) 2003 San Francisco (USA). Citeseer.
Reddy, R., France, R., Ghosh, S., Fleurey, F. & Baudry, B. Model composition-a signature-based approach. Workshop on Aspect Oriented Modeling (AOM@MODELS) 2005 Montego Bay (Jamaica). 264-278.
Rivera, J. E. & Vallecillo, A. 2008. Representing and operating with model differences. *Objects, Components, Models and Patterns*. Springer.
Schürr, A. Specification of graph translators with triple graph grammars. International Workshop on Graph-Theoretic Concepts in Computer Science, 1995 Aachen (Germany). Springer, 151-163.
Shashank, S. P., Chakka, P. & Kumar, D. V. A systematic literature survey of integration testing in component-based software engineering. International Conference on Computer and Communication Technology (ICCCT), 2010 Andhra Pradesh (India). IEEE, 562-568.
Shvaiko, P. & Euzenat, J. 2013. Ontology matching: state of the art and future challenges. *IEEE Transactions on Knowledge and Data Engineering (TKDE),* 25**,** 158-176.
Spaccapietra, S. & Parent, C. 1991. Conflicts and correspondence assertions in interoperable databases. *ACM SIGMOD Record,* 20**,** 49-54.
Steinberg, D., Budinsky, F., Paternostro, M. & Merks, E. 2009. *EMF: Eclipse Modeling Framework*, Addison-Wesley.
Treude, C., Berlik, S., Wenzel, S. & Kelter, U. Difference computation of large models. European Software Engineering Conference (ESEC@FSE), 2007 Dubrovnik (Croatia). ACM, 295-304.
Vallecillo, A. 2010. On the combination of domain specific modeling languages. *Modelling Foundations and Applications*. Springer.
Van Den Brand, M., Protić, Z. & Verhoeff, T. Fine-grained metamodel-assisted model comparison. International Workshop on Model Comparison in Practice (IWMCP), 2010 Malaga (Spain) ACM, 11-20.
Varró, D. & Balogh, A. 2007. The model transformation language of the VIATRA2 Framework. *Science of Computer Programming,* 68**,** 214-234.
Vogel, L. 2014. *Contributing to the Eclipse Project: Principles, Plug-ins and Gerrit Code Review*, Lars Vogel.
Voigt, K. & Heinze, T. 2010. Metamodel matching based on planar graph edit distance. *Theory and Practice of Model Transformations*. Springer.
Voigt, K., Ivanov, P. & Rummler, A. Matchbox: combined meta-model matching for semi-automatic mapping generation. ACM Symposium on Applied Computing (SAC), 2010 New York (USA). ACM, 2281-2288.

Voormann, H. 2014. *Luna Rising* [Online]. Available: http://eclipsehowl.wordpress.com/2014/06/25/luna-rising/ [Accessed November 2014].

Wagelaar, D., Iovino, L., Di Ruscio, D. & Pierantonio, A. 2012. Translational semantics of a co-evolution specific language with the EMF transformation virtual machine. *Theory and Practice of Model Transformations*. Springer.

Wenzel, K. Ontology-Driven Application Architectures with KOMMA. International Workshop on Semantic Web Enabled Software Engineering (SWESE), 2011 Seiten (Germany). 78-85.

Whittle, J., Hutchinson, J. & Rouncefield, M. 2014. The state of practice in model-driven engineering. *IEEE software,* 31**,** 79-85.

Wirth, N. 1996. Extended Backus-Naur Form (EBNF). *ISO/IEC,* 14977**,** 2996.

Wooldridge, M. 1997. Agent-based software engineering. *IEEE Transactions on Software Engineering (TSE),* 144**,** 26-37.

Wouters, L. 2013. *Multi-Domain Expert-User Modeling Infrastructure*. Université Paris X.

Wouters, L. & Gervais, M.-P. xOWL: An Executable Modeling Language for Domain Experts. IEEE Enterprise Distributed Object Computing Conference (EDOC), 2011 Helsinki (Finland). IEEE, 215-224.

Xing, Z. & Stroulia, E. UMLDiff: an algorithm for object-oriented design differencing. International Conference on Automated software Engineering (ASE), 2005 Long Beach (USA). ACM, 54-65.

Zito, A., Diskin, Z. & Dingel, J. 2006. Package merge in UML 2: Practice vs. theory? *Model Driven Engineering Languages and Systems*. Springer.

ANNEXE A : ALGORITHME DE REPRODUCTION

L'algorithme se fonde sur une représentation des correspondances sous la forme suivante :

C :	R	E1	E2	…	En

Où R est le type relation et E1, E2,.., En, les éléments de (méta)modèles.

```
Variable D en Numérique//Nombre de HLC
Variable S en Numérique //Nombre d'éléments
Tableau HLC [D] [S] en Caractère // Les correspondances du niveau M2
Tableau LLC [ ] [ ] en Caractère // Les correspondances du niveau M1
Tableau allElements [ ] [ ] en Caractère
Tableau tmp [ ] en Caractère
Variable k, l en Numérique

Début
Variables i, j en Numérique
l <- 0
Pour i <- 0  à D
allElement[0][i]= HLC [i][j] //le même type de relation du HLC est dupliqué
        Pour j<-1 j à S
                allElements[j][] = InstanceOf(HLC[i][j])
                //Chaque ligne contient les instances d'un méta-élement.
        j suivant
    Reproduction(allElements, tmp, 0)
    Vider(allElement) //Supprimer les éléments du tableau
    Vider(tmp)
i suivant
Fin

Fonction : Reproduction(Tableau tabElements [ ][ ], Tableau tmp[ ], k)
Variables i, j en Numérique
Si (k == nbLigne (tabElements [ ][ ])
{
    i <- 0
    Tant que (i<k)
    {
        LLC[l][i]=tmp[i]
        i <- i+1
        l <- l+1
    }
}
Sinon
```

```
{
    j <-0
    Tant que (j < nbColonne (tabElements [k] [ ]))
    {
        tmp [k] = tabElements [k][j]
        Reproduction(tabElements, tmp, k+1)
    }
}
Renvoyer LLC
Fin fonction
```

ANNEXE B : GRAMMAIRE XTEXT DE L'ÉDITEUR TEXTUEL

Cette annexe présente la grammaire complète de la notation textuelle, dans le formalisme pseudo-EBNF de XTEXT, et précise les mots clés utilisés pour chaque concept, relation, et référence entre concepts.

```
grammar com.irit.maco.hmcs.m2c.M2CDSL with org.eclipse.xtext.common.Terminals

import "http://IRIT.HMCS.POC/MMC"
import "http://www.eclipse.org/emf/2002/Ecore" as ecore
import "http://www.eclipse.org/emf/2003/XMLType" as type

Packages returns Packages:
'Packages'
name=EString
(gathersM+=RefModel ("," gathersM+=RefModel)*)
(gathersC+=CorrespondenceModel ("," gathersC+=CorrespondenceModel)*);

RefModel returns RefModel:
   'import'
   'model' ref=EString
   '{' dispose+=RefElement ("," dispose+=RefElement)* '}';

CorrespondenceModel returns CorrespondenceModel:
   'create'
   name=String2
   'define' gathersC+=Correspondence ("," gathersC+=Correspondence)* ';';

Correspondence returns Correspondence:
   'Correspondence' name=EString
   (mandatory?='mandatory')?
   'basedOn' gathersR=Relationship
   ('weight=' weight=EInt)?;

EString returns ecore::EString:
   STRING | ID;

String2 returns type::String:
   TYPE;
```

```
terminal TYPE:
   "M2C" | "M1C";

Relationship returns Relationship:
   Similarity_Impl | Aggregation_Impl | Generalization | Dependency_Impl | Play |
   Requirement | Contribution;

RefElement returns RefElement:
   {RefElement}
   ('RefElement' ref=EString)?
   ('sourceRel' '(' sourceRel+=[Relationship|EString] ("," 
   sourceRel+=[Relationship|EString])* ')')?
   ('targetRel' '(' targetRel+=[Relationship|EString] ("," 
   targetRel+=[Relationship|EString])* ')')?;

Expression returns Expression:
   {Expression}
   '='
   '['
   ('Language' '=' Language=EString)? ','
   ('Body' '=' Body=EString)?
   ']';

Similarity_Impl returns Similarity:
   name='Similarity'
   'Between'
   '(' sourceElt+=[RefElement|EString] ("and" sourceElt+=[RefElement|EString])* ')'
   (',' '(' targetElt+=[RefElement|EString] ("and"
targetElt+=[RefElement|EString])* ')')?
   'with' ':'
   (bidirectional?='bidirectional')?
   ('Expression' possesses=Expression)?;

Aggregation_Impl returns Aggregation:
   name='Aggregation'
   'Between'
   '(' sourceElt+=[RefElement|EString] ("and" sourceElt+=[RefElement|EString])* ')'
   (',' '(' targetElt+=[RefElement|EString] ("and"
targetElt+=[RefElement|EString])* ')')?
   'with' ':'
   (bidirectional?='bidirectional')?
   ('Expression' possesses=Expression)?;

Generalization returns Generalization:
   name='Generalization'
   'Between'
   '(' sourceElt+=[RefElement|EString] ("and" sourceElt+=[RefElement|EString])* ')'
   (',' '(' targetElt+=[RefElement|EString] ("and"
targetElt+=[RefElement|EString])* ')')?
   'with' ':'
   (bidirectional?='bidirectional')?
   ('Expression' possesses=Expression)?;

Contribution returns Contribution:
   name='Contribution'
   'Between'
   '(' sourceElt+=[RefElement|EString] ("and" sourceElt+=[RefElement|EString])* ')'
   (',' '(' targetElt+=[RefElement|EString] ("and"
   targetElt+=[RefElement|EString])* ')')?
```

```
   'with' ':'
   (bidirectional?='bidirectional')?
   ('Expression' possesses=Expression)?;

Dependency_Impl returns Dependency:
   name='Dependency'
   'Between'
   '(' sourceElt+=[RefElement|EString] ("and" sourceElt+=[RefElement|EString])* ')'
   (',' '(' targetElt+=[RefElement|EString] ("and"
targetElt+=[RefElement|EString])* ')')?
   'with' ':'
   (bidirectional?='bidirectional')?
   ('Expression' possesses=Expression)?;

Requirement returns Requirement:
   name='Requirement'
   'Between'
   '(' sourceElt+=[RefElement|EString] ("and" sourceElt+=[RefElement|EString])* ')'
   (',' '(' targetElt+=[RefElement|EString] ("and"
   targetElt+=[RefElement|EString])* ')')?
   'with' ':'
   (bidirectional?='bidirectional')
   ('Expression' possesses=Expression)?;

Play returns Play:
   name='Play'
   'Between'
   '(' sourceElt+=[RefElement|EString] ("and" sourceElt+=[RefElement|EString])* ')'
   (',' '(' targetElt+=[RefElement|EString] ("and"
   targetElt+=[RefElement|EString])* ')')?
   'with' ':'
   (bidirectional?='bidirectional')?
   ('Expression' possesses=Expression)?;

EInt returns ecore::EInt:
   '-'? INT;
```

ANNEXE C : LISTE DES ABRÉVIATIONS

- AMT : Assisted Matching Tool
- CMS : Conference Management System
- CMT : Consistency Management Tool
- CRE : Compte Rendu d'Examen
- DIR : Domain Independent Relationship
- DSR : Domain Specific Relationship
- HLR : High Level Relationship
- HMCS : Heterogeneous Matching and Consistency management Suite
- LLR : Low Level Relationship
- MFT : Model Filtering Tool
- MT : Matching Tool
- OFT : Ontology Filtering Tool
- RT : Refining Tool
- SE : Semantic Expression
- SED : Semantic Expression DSL
- SU : Service d'Urgence